



®

FreeBSD[®] **JOURNAL**

September/October 2020

Contributing Onboarding Issue

Code Review

Writing Good Commit Messages

How to Contribute Without Being a Programmer

How to Become a Doc Committer

Google Summer of Code

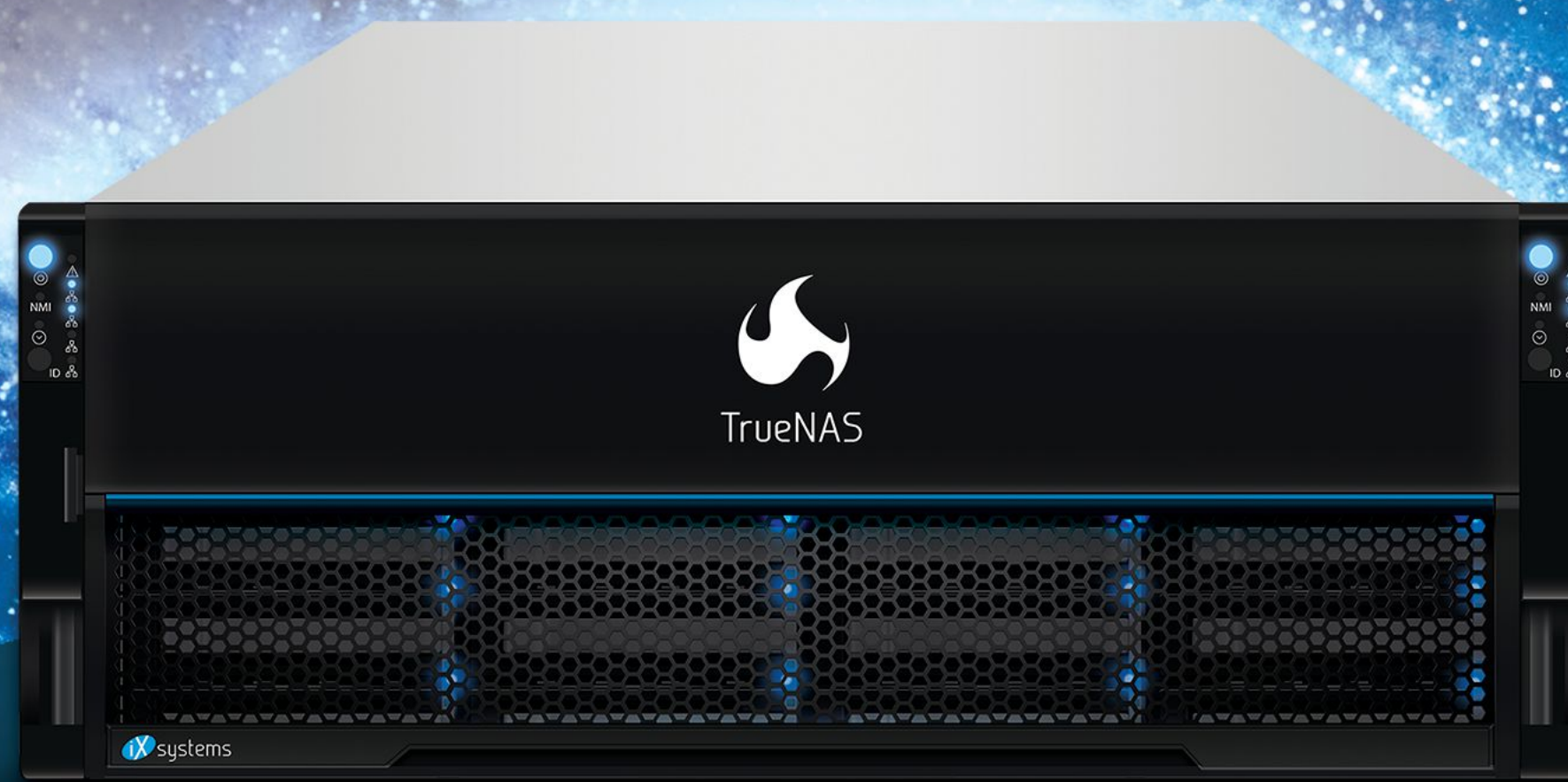
Writing an Article for the FreeBSD Journal

Why do you use FreeBSD?

Interview: Warner Losh, Part 2

TrueNAS® M-SERIES
Powerfully Scalable Enterprise Storage

OPEN STORAGE FOR ENTERPRISE WORKLOADS



UTILIZES FLASH-OPTIMIZED ZFS TECHNOLOGY
IDEAL FOR LATENCY-SENSITIVE AND BUSINESS-CRITICAL
VIRTUAL MACHINES AND PHYSICAL WORKLOADS.

**PERFORMANCE AND SCALE
WITHOUT COMPROMISE**

**INTELLIGENT STORAGE
OPTIMIZATION**

**SELF-HEALING DATA
PROTECTION**

**UNLIMITED SNAPSHOTS AND
REPLICATION**

Contact iXsystems to Learn More about what TrueNAS® can do for your business!

ixsystems.com/TrueNAS | (855) GREP-4-iX



FreeBSD[®] JOURNAL

Editorial Board

- John Baldwin • FreeBSD Developer and Chair of FreeBSD Journal Editorial Board.
- Justin Gibbs • Founder of the FreeBSD Foundation, President of the FreeBSD Foundation, and a Software Engineer at Facebook.
- Daichi Goto • Director at BSD Consulting Inc. (Tokyo).
- Tom Jones • FreeBSD Developer, Internet Engineer and Researcher at the University of Aberdeen.
- Dru Lavigne • Author of *BSD Hacks* and *The Best of FreeBSD Basics*.
- Michael W Lucas • Author of *Absolute FreeBSD*.
- Ed Maste • Director of Project Development, FreeBSD Foundation and Member of the FreeBSD Core Team.
- Kirk McKusick • Treasurer of the FreeBSD Foundation Board, and lead author of *The Design and Implementation* book series.
- George V. Neville-Neil • Director of the FreeBSD Foundation Board, Member of the FreeBSD Core Team, and co-author of *The Design and Implementation of the FreeBSD Operating System*.
- Philip Paeps • Secretary of the FreeBSD Foundation Board, FreeBSD Committer, and Independent Consultant.
- Kristof Provost • Treasurer of the EuroBSDCon Foundation, FreeBSD Committer, and Independent Consultant.
- Hiroki Sato • Director of the FreeBSD Foundation Board, Chair of Asia BSDCon, Member of the FreeBSD Core Team, and Assistant Professor at Tokyo Institute of Technology.
- Benedict Reuschling • Vice President of the FreeBSD Foundation Board and a FreeBSD Documentation Committer.
- Robert N. M. Watson • Director of the FreeBSD Foundation Board, Founder of the TrustedBSD Project, and University Senior Lecturer at the University of Cambridge.
- Mariusz Zaborski • FreeBSD Developer, Manager at Fudo Security.

S&W PUBLISHING LLC

PO BOX 408, BELFAST, MAINE 04915

- Publisher** • Walter Andrzejewski
walter@freebsdjournal.com
- Editor-at-Large** • James Maurer
jmaurer@freebsdjournal.com
- Copy Editor** • Annaliese Jakimides
- Design & Production** • Reuter & Associates
- Advertising Sales** • Walter Andrzejewski
walter@freebsdjournal.com
Call 888/290-9469

FreeBSD Journal (ISBN: 978-0-615-88479-0) is published 6 times a year (January/February, March/April, May/June, July/August, September/October, November/December).

Published by the FreeBSD Foundation,
3980 Broadway St. STE #103-107, Boulder, CO 80304
ph: 720/207-5142 • fax: 720/222-2350
email: info@freebsd.foundation.org

Copyright © 2020 by FreeBSD Foundation. All rights reserved. This magazine may not be reproduced in whole or in part without written permission from the publisher.

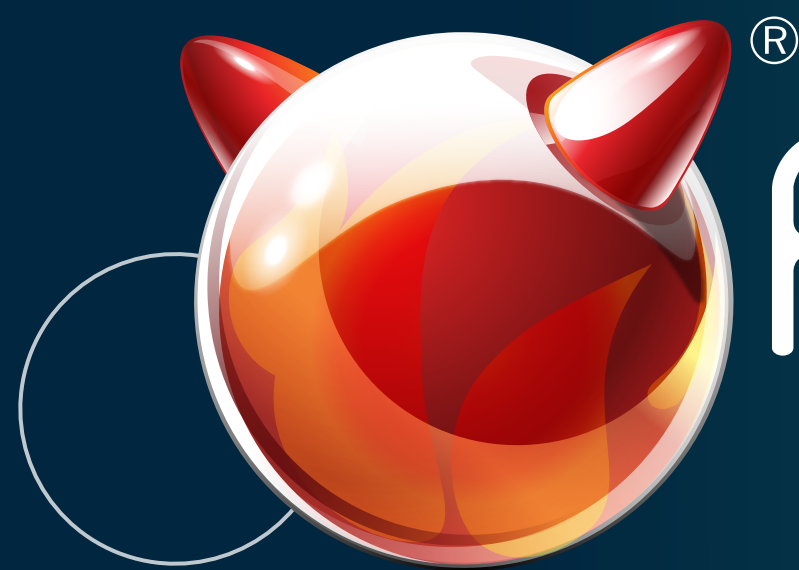
LETTER

from the Foundation

Welcome to the September/October 2020 issue of the *FreeBSD Journal*. The issue focuses on Contributing to FreeBSD. It covers lots of paths to joining the Project, and also talks more about why people use FreeBSD in the first place. The Foundation team is pretty excited about this issue because it aligns really closely with the advocacy part of our mission. In the Before times, we traveled all over the world, introducing folks to FreeBSD and helping them get started contributing to the Project. Thankfully, due to the generosity of our community, we're able to continue spreading the word to this day. While we might not be at conferences, we're still holding workshops at online events. Our new FreeBSDFridays series of one-hour FreeBSD 101 classes is going strong, and we're about to hold our first online FreeBSD Vendor Summit, November 11-13, 2020. Vendor Summits are great because they offer companies who use FreeBSD the chance to talk directly to developers to get their questions answered and needs met. This year's event is particularly exciting not only because it is open to anyone interested in attending, but also because it's free.

The Foundation continues to work with members of the community to help streamline the onboarding process, and *Journal* issues like this one are just part of the plan. Please take a moment to check out the issue, and be sure to share with those who may be interested in contributing to FreeBSD!

On behalf of the FreeBSD Foundation,
Anne Dickison



11 Interview: Warner Losh, Part 2

By Allan Jude and Benedict Reuschling

Special Section on Contributing/Onboarding

16 Code Review *By John Baldwin*

19 Writing Good Commit Messages *By Ed Maste*

23 How to Contribute Without Being a Programmer *By Danilo Baio*

29 How to Become a Doc Committer *By Benedict Reuschling*

32 Google Summer of Code *By Mariusz Zaborski*

35 Writing an Article for the *FreeBSD Journal* *By Roller Angel*

37 Why do you use FreeBSD? *By Anne Dickison*

3 Foundation Letter *By Anne Dickison*

5 New Faces of FreeBSD *By Dru Lavigne*

8 We Get Letters *By Michael W Lucas*

39 Events Calendar *By Anne Dickison*

new faces

of FreeBSD

BY DRU LAVIGNE

This column aims to shine a spotlight on contributors who recently received their commit bit and to introduce them to the FreeBSD community.

*In this installment, the spotlight is on **Rainer Hurling**, who received his ports bit in August, and **Gordon Bergling** who received his doc bit in June.*

.....
Tell us a bit about yourself, your background, and your interests.

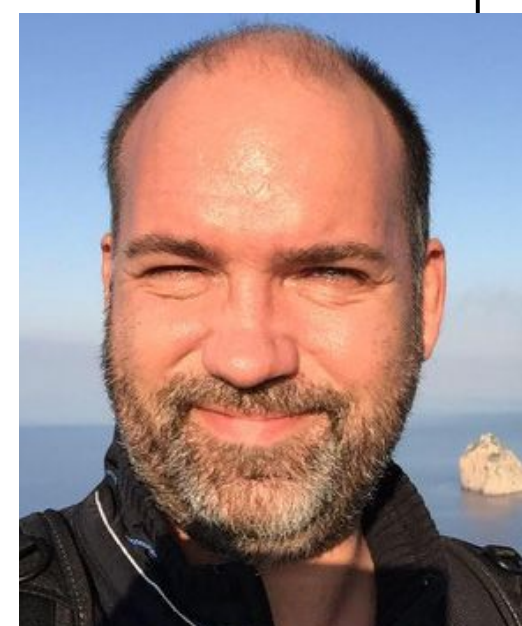
• **Rainer:** I am Rainer Hurling and I grew up in the north of Germany. Since completing my studies, I have lived in the middle of Germany, near Göttingen. I have been a computer enthusiast since I was a teenager. After my first experiences in 1977 during a school internship with a Siemens TR 440 mainframe, I got an increased understanding for hardware and software by working with a homemade computer based on a SC/MP CPU. In the early 1980s, I got my first real computer system, a Eurocom II board with 6809 CPU from Eltec Elektronik, Mainz, Germany. The included FLEX OS from TSC came with an assembler, a disassembler, Pascal, Forth, and Fortran (and Sargon Chess, of course), all 8-inch floppy disks. In 1983, an Apple IIe was added, on which I programmed mainly with UCSD Pascal.



Because of my second passion, the natural sciences—especially forest ecosystems—I studied forestry and sometime after that I had the opportunity to do a doctorate on a topic in forest protection. For almost 20 years now, I have been working as a head of section and chief scientist in the department of forest protection of a forestry research institute in Göttingen. We intensively work with several packages of the R Project for Statistical Computing, we use PostgreSQL databases with PostGIS, and, in particular, geo-based software like SAGA GIS, QGIS, and OpenJUMP. In our department I am one of the persons who methodically drives data management and data analysis and the one who also tries to contribute to these projects. Among others, I have been the maintainer of the ports math/saga, graphics/qgis, graphics/openjump, and databases/mdbtools for years now. One of my personal goals is to help keep this geo-based and mathematical software running on FreeBSD.

In private life, I am also interested in astrophysics and try to improve existing software there as well as the maintainer of astro/astrometry and astro/py-ephem. Since 1981, I have had an amateur radio license (DH6BAG). It is nice to watch how the bundle of software for ham radio grows and prospers on FreeBSD.

• **Gordon:** I am currently a freelance software developer based in Leipzig, Germany. I have worked in the IT industry since the late 1990s and have experience in a variety of industry fields ranging from e-commerce, pharmacy, to finance. My work is focused on distributed systems and business applications. I am very interested in IoT and edge computing, but I haven't found the time yet to look at these things in a real-world situation. Besides



software development, I am also interested in photography and anything related to music. I love traveling and getting to know new cultures and people.

.....

How did you first learn about FreeBSD and what about FreeBSD interested you?

• **Rainer:** During my time as a doctoral student at the university in the mid-'90s, the Windows operating system was barely usable for fast, extensive scientific tasks on a PC. Instead, I used OS/2 Warp4 for several years, and as an IBM registered developer and tester, I contributed a little bit. Unfortunately, OS/2 was not further developed and an alternative had to be found.

In 1996, I read about this free, driven-by-demons, UNIX-like system and decided to try FreeBSD, at that time, version 2.0.5. It was like a rebirth. The same hardware that was used with Windows and OS/2 before now behaved performant and stable. At that time, there were already countless programs available. From 1997 on, I used the software R on FreeBSD and began to use FreeBSD as a scientific desktop, running all my requirements for a scientific workplace as open source on a Window Maker desktop.

From the beginning, I loved the clarity and structure of FreeBSD and also the very complete documentation. Very early, I decided to use CURRENT and with it the latest available adaptations. FreeBSD mailing lists, forums, and IRC have been invaluable over the years. There was always fast, friendly, and competent help.

• **Gordon:** I was fortunate that one of my classmates was into FreeBSD and that he gave me a CD of FreeBSD 4.0-RELEASE when it was just released. I started to use FreeBSD at work as a firewall and for some web server and IDS purposes. Later, I started to use FreeBSD as my main operating system for a couple of years. I always liked the completeness of the operating system, ranging from the base system to the kernel and, of course, the ports collection.

.....

How did you end up becoming a committer?

• **Rainer:** In the early years of using FreeBSD, I posted some suggestions in mailing lists and forums every now and then. The creation of official bug reports resulted from a desire to submit software not yet available as a new port. As one of my first submissions, I submitted a PR in 2009 for a new port: math/saga. After some off-list communication, the port was accepted just three days later. What a feeling! About 220 PR submits followed over the next years. In 2014, I was asked by Wen Heping if I was interested in a commit bit. Although I was very happy with the invitation, my job was so demanding that I declined. In the time that followed, I had to postpone such attractive offers twice more.

This year, in late spring, Tobias C. Berner asked if I was interested in a ports commit bit. After some consideration, I happily accepted. Tobias and Gleb Popov are really nice, patient, and helpful mentors, and it has been great fun to learn from them.

• **Gordon:** had contributed some patches from time to time around 2004. After 2018, I had some free time and remembered how much fun the Project was in the old days, looked through some release documentation and installed it again. I always liked the comprehensive documentation, especially the man pages and the FreeBSD Handbook, so I started to submit some patches and finally got my commit bit at the beginning of June this year.

.....
How has your experience been since joining the FreeBSD Project? Do you have any advice for readers who may be interested in also becoming a FreeBSD committer?

- **Rainer:** In my few weeks as a ports committer, I have gained experience with subversion, learned many conventions of the commit process, and extended my general understanding of the ports framework. Compared to many other communities, I find the FreeBSD community to be particularly open, helpful, fast, and concrete. It is a pleasure to work on solutions with others in this community.

Even as a submitter without professional programming or network administration knowledge, there is a lot you can contribute in the areas of ports and docs. But I have very little experience as a committer so far. Nevertheless, I recommend interested people to take responsibility for further developments in the FreeBSD ports. As a submitter, you should begin with a particular area that interests you.

- **Gordon:** The experience has been just great. Benedict Reuschling is my mentor, and he does a great job of explaining the nuances of quality in documentation and he always offers a helping hand when I run into problems. If someone is contributing patches via bugzilla, they should also provide a differential in FreeBSD Phabricator. That way, it will be more likely to be reviewed and picked up for a commit later. FreeBSD developers will notice your number of submitted patches and that might lead to a commit bit.

DRU LAVIGNE is the author of *BSD Hacks* and *The Best of FreeBSD Basics*.

Thank you!

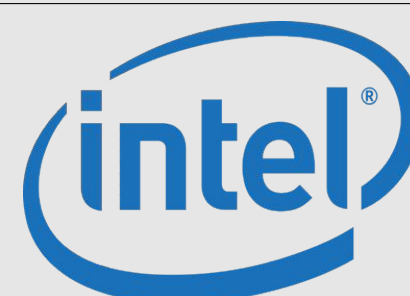
The FreeBSD Foundation would like to acknowledge the following companies for their continued support of the Project. Because of generous donations such as these we are able to continue moving the Project forward.



Are you a fan of FreeBSD? Help us give back to the Project and donate today! freebsd.foundation.org/donate/

Please check out the full list of generous community investors at freebsd.foundation.org/donors/

Uranium



Iridium



Platinum

NETFLIX

Gold



Silver

BECKHOFF



STORMSHIELD





**Oh Fiercely Fearless Feckless Letters Entity,
I've used open-source software like FreeBSD for
ages now. It's improved my life and I want to give
back. But I can't seem to find a way into a project.
Either my contributions are ignored or my skills
don't apply or my work is rejected. I feel totally
stuck on the outside. How can I help my favorite
project?**

—Eager But Baffled

Dear EBB,

I can't decide if you thought I wouldn't know what the word "feckless" meant or if you thought I'd recognize its truth and go with it. For the record, it's the latter.

I suspect you've fallen victim to Sysadmin Rule #17: You are solving the wrong problem.

It's easy to look at a major open-source project and say, "Wow, oooh, I want to do that." It's also easy to look at professional skateboarders, rugby players, and cage fighters and say the same. You want to become a particular sort of major player. The vision of yourself as a digital version of Samuel L. Jackson or Jason Statham might delight you, but the fantasy overlooks a tiny, minor, minuscule detail.

Becoming a major player takes a lot of work.

Skateboarders spend years faceplanting before they can do those fancy tricks. Cage fighters get punched and kicked and wedgied for years, nonstop, before winning their bouts. And let's not discuss rugby.

The major players of technology achieved that skill level by spending decades getting face-punched by compiler errors, kicked by bugs in hobnailed boots, and deluged by petulant, ineffective, and downright nonsensical technology changes. Many folks who work at the peaks of open source have additional honorifics after their name, like "PhD" or "Snobby McSnooty Prize for Computing" or "hacker or hackers unknown."

Are you on the couch watching rugby and wishing you could do that? Or are you spending each day lifting weights, running marathons, and thwacking your tenderest anatomy with a ball peen hammer before each meal to get ready to join the pub team and grind your way up?

Ask yourself that question. Then decide which of three paths to take.

The first, "give up," is unworthy of my time and attention. You are fully qualified to implement this plan without my guidance.

Solution two would be to change yourself. If you want to be the next Major Player of FreeBSD, do the work to become that. Can you program in C? If not, learn that. Yes, fancy languages like Perl and Smalltalk and Fortran get all the glory, and the kids like some weird thing named after a snake, but the project's language is C, and they're not going to change to fit

your prejudices. If you want to develop a Haskell kernel, join a different project. I'm sure you'll have many to choose from.

The best way to learn a programming language is to want to accomplish a task in that language.

A smart person would start with the bug database. Look for bugs that you can replicate with your system. Replicate them, contact the reporter for more information, and start digging. Develop patches that resolve the problem. You'll earn the respect of the community.

Maybe you don't want to clean up someone else's bugs. You want to do something big and world changing. Large open-source projects are not about writing exciting new components from scratch, but rather about slow evolution. Major projects happen, but they go to respected people in the community. You can try it, though.

Back in a previous century, early in my systems administration career, I found that I needed an FTP client that could do recursive fetches. While three or four FTP clients existed—and half of them even worked, more or less—my life would have been much simpler if the client included in FreeBSD 2.x had that feature. I wanted to hone my knowledge of C. I'd read the first edition of Kernighan and Ritchie's *The C Programming Language* and had written a few petty buffer overflows disguised as software. The cliché was that open source was about scratching itches—well, I had the itch, I should be the one to scratch it.

I started reading source code.

And rereading source code.

Then I printed the source code for `ftp(1)`, taped it on my home office wall, and went at it with several colors of highlighters, red and green pens, marking sections of code that my inadequate brain thought were important or relevant and connecting disparate functions together. It was like one of those conspiracy theorists' corkboards all tied together with string, except it was demonstrably real and every time I touched any bit of it the whole thing imploded.

Had I taken the time to ask any FreeBSD committer about my little project, they would have steered me to something more realistic, like juggling saber-toothed porcupines. Or at least advised me to wear protective gear against the inevitable ball-peen hammers. I learned vast amounts about C. I learned how to handle core dumps. I learned exactly how inadequate my programming skills were.

I could have become a programmer. All I had to do was keep pounding away at the problem.

After several months of bending my brain, I removed everything except the printouts from the office and moved out of that house, in the hope that the new tenants would be wiser than myself and stay off the Internet forever.

I chose the third way.

You have skills in something, presumably. In your years, you've continued breathing and maintained essential digestive functions. When you consider that most of the people who have ever lived are now dead, you're doing pretty well.

Take the thing that you do and do it for your project.

Maybe you like helping people with problems you've already solved. Every project has formal or informal support channels, mailing lists, forums, whatever. Hang out on those forums, helping people.

You know who gets plaudits and kudos from developers? Testers.

Before release day, download the release-candidate install media. Make a checklist of all installer options, and methodically test them all. Report errors.

If for some deranged reason you want something more programmery, learn how to write and use software tests.

If your job involves some weird edge of the software, blog about your experiences in that edge. Or scribble some notes for the wiki. Perhaps you're among the truly unfortunate and can write. No further action is necessary on your part at this time, but don't worry. The cabal will be by shortly with tasers and nets. Don't struggle. It won't help.

The point is: do the thing you do. But do it for FreeBSD.

It's okay to change what you do. Time spent helping other users makes people's lives better. Time spent documenting how you can tune a filesystem but you can't tuna fish does the same. The time I spent methodically studying source code improved my code reading skills, which I rely on today. On a community level, people will remember you.

Specifically, they'll remember the quality of your work. They will treat you with the seriousness that the quality of your previous work merits.

You can't wedge your way into a community. Open source doesn't work that way. It's not a block party where the right mailing address grants you admission. The only way to gain admission to an open-source community is by completing quality work. If you can do that, you won't have to fight to gain entrance. More and more community members will tie social strings to you until you find yourself sentenced to life in that community. People will bribe you into taking on the community's absolute worst jobs, simply because you have a reputation for completing that kind of task successfully.

What kind of horrid jobs? You know, like answering a letters column for the community journal.

Have a question for Michael?
Send it to letters@freebsdjournal.org

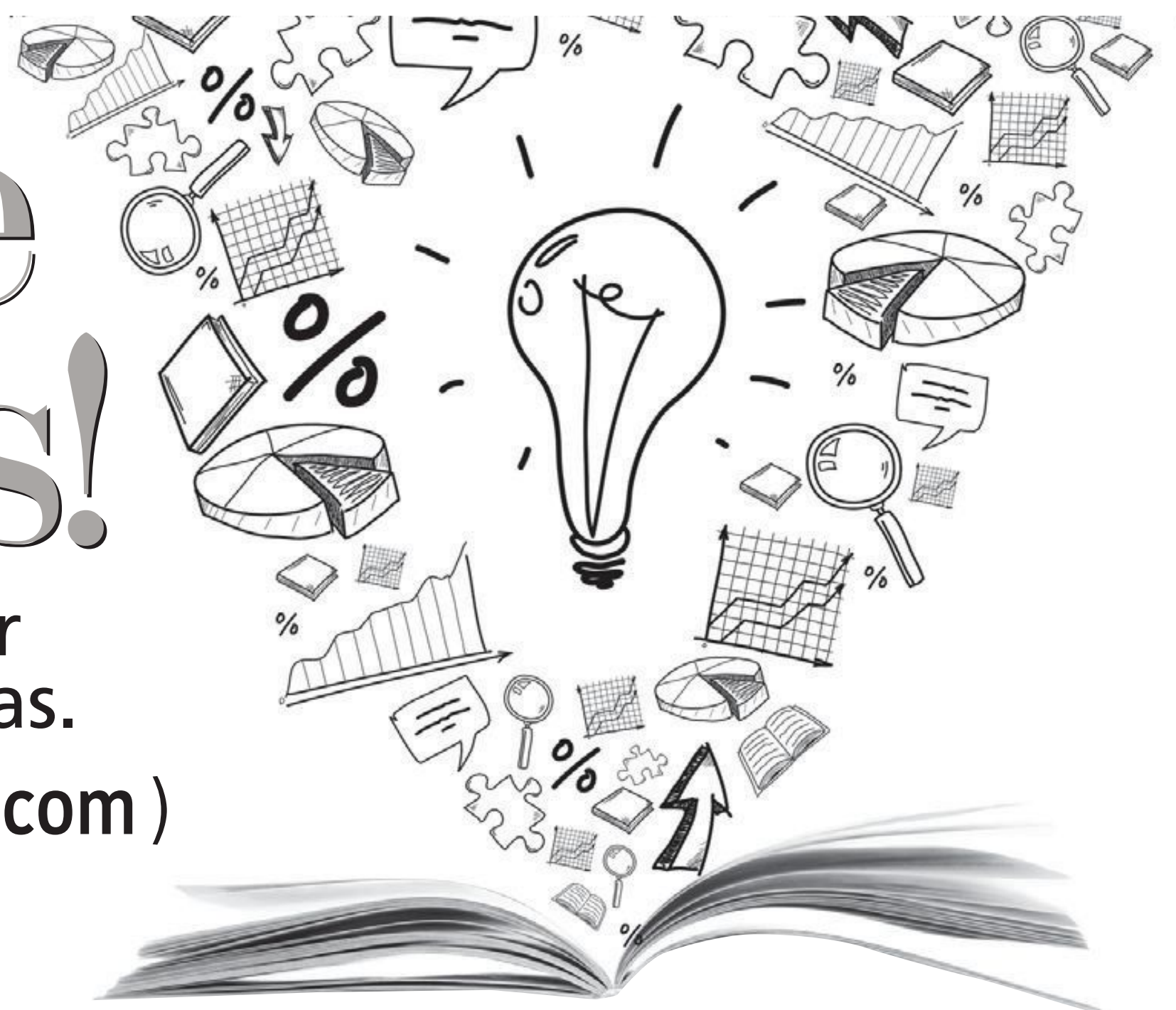


MICHAEL W LUCAS (<https://mwl.io>)'s newest books are *SNMP Mastery* and *Terrapin Sky Tango*.

Write For Us!

Contact Jim Maurer
with your article ideas.

(jmaurer@freebsdjournal.com)



AN • INTERVIEW • WITH Warner Losh part 2

• • • • • by Allan Jude and Benedict Reuschling

BSD Now Episode 362, 2.11 BSD Restoration. Recorded for August 5, 2020.

Benedict Reuschling and Allan Jude speak with Warner Losh about Unix history and its interesting tidbits. They discuss his 2.11 BSD Restoration Project, of which the Unix Heritage Society is part. And what's that thing he wrote called devmatch? We hope you'll enjoy it all! Part 1 appeared in the July-August 2020 issue.

REUSCHLING: Let's return to the Unix Heritage Society you mention in your blog and talks—what is it exactly? How can people participate?

WARNER LOSH: The Unix Heritage Society is sometimes called TUHS because the full name is a mouthful. They preserve the different Unix artifacts that have existed over the years inside Bell Labs and also once Unix was out in the world. TUHS has built an archive that's freely available through their website, and they strive to not only preserve the artifacts of Unix history in terms of code and binaries, but they also seek out old troves of papers about Unix. For example, when someone retires or dies and the family donates the person's papers, they scan them and make them available. Anyone who is interested can discover what it was like when some of the historical developments were actually happening, and read about them in a larger context.

One example is a group of papers about networking at Bell Labs that aren't TCP/IP. There's something called data kit and it represents a mindset at the Labs. They were a phone company, and they had a circuit-switch mentality because that's how phone calls are created. You create a circuit from the originator to the other side, and it's hardwired the whole time. So, their networking technology—their data-networking technology—reflected that. You would create a circuit, and then you would send data over the circuit. And that's a very different mindset from the packet-switch paradigm that we have today and that ultimately wound up winning. In the Unix Heritage Society collections, you can read about this and dozens of other historical events or mindsets or cultural happenings.

If you've got bits of lost history, or anything that's old and Unix related, then that's potentially of interest. There is both a public and a private archive for TUHS. William Toomey is the person who started it, and he maintains a private archive of early commercial releases on which the copyrights have not as yet expired, but he archives against the day that he can release them.

So even if you have something that can't be widely distributed, you can still deposit it in the archive so that it won't be lost to history. There are different ways to participate, but, generally, being on the mailing list, taking part in the conversations, following the different discussions, and participating in projects that come up every year or two as new artifacts surface are the primary ways most people get involved.

JUDE: That's really interesting. You just wonder how many things are out there but are lost or just undiscovered and how we can make sure more of the stuff we create doesn't end up getting lost.



LOSH: That's right. There are also other efforts. Bitsavers goes in and scans documents. They look at old computer manuals from DEC and Sun and other companies, and they'll scan those—particularly the ones that help people write simulators. They also preserve different magazine articles, and recently somebody donated all the early DECUS newsletters. DECUS is a DEC users' group that held symposium conferences back in the day, published proceedings from them, and produced newsletters that they would send out monthly to the members, talking about different developments.

It's a good thing, because although we have received the wisdom from the past, we don't always know how that wisdom accumulated or what bits of knowledge were retained or discarded. Discovering that is really exciting; at least it is to me, and given the popularity of some of the subjects I've written about, maybe it interests a few other people too.

JUDE: I think we need to give more consideration to preserving the history of not just the work we're doing but more of our thinking around it because nowadays people don't write papers and produce proceedings quite as much as they did before. A recording of a talk from BSDCan or whatever is good, and it is something we're starting to preserve, but it doesn't necessarily always get into why things were done a certain way—which could be interesting questions down the road.

LOSH: Exactly. When you read a paper, you have a different audience from when you're giving a talk. And the talk has a time constraint, and you're talking to people who have a wide range of abilities and interests and so you try to keep the talk interesting and relevant in that context. You put different things in it than you do a paper where you want to say, I did this, this, and this, and you don't want to leave anything out—even if it bores your reader a little—because some reader, sometime in the future, will want to know—is that why this crazy thing is going on now? I much like the approach of preserving all the different forms of media that are produced.

One thing that worries me is we have mailing list archives, and we have forum archives—kind of. But the forums tend to be more ephemeral and come and go. As a result, a lot of the thinking gets lost. And it's a slow erosion over time. If you're talking about how to set the jumpers on an iCOBUS network card, nobody's really gonna think to preserve that. But if you are trying to recreate a FreeBSD 4 system and realize, oh, I need period hardware, or I need a network card, or, this is all I can find, then that sort of thing suddenly becomes more interesting. And maybe a FreeBSD 1 system would need that, whereas, with FreeBSD 4, everything's plug and play. But that's the kind of thing that you might not think of when you discard it. You might think that's unimportant and that you don't need that—but others might.

JUDE: That's interesting. When I was working on the encrypted boot loader stuff, one of the papers I spent the most time on was getting into how FreeBSD boots. You recently wrote an article on how you actually basically reboot or shut down the machine.

REUSCHLING: Properly.

LOSH: Safely.

JUDE: So, what is the proper way to restart a Unix machine?

LOSH: Well, on modern systems it's like shutdown. Shutdown will make sure everything gets shut down and will try very hard to make sure all the dirty buffers get written out. But it will also bound its efforts so that it won't try forever to write things out, and so the system will actually reboot. It's kind of unsatisfying. You can't just flip a machine off. On some systems, you can type `synch` and then turn it off immediately, because the `synch` is synchronous. Linux has a synchronous `synch`. When `synch` returns, everything has been written. If the system is otherwise quiet, you could turn it off, for example.

BSD just schedules the writes. And there are some complications with UFS soft updates where, when a buffer is written, it will re-dirty itself so that it gets written again when other data is written out, and so that it can present a consistent view of the media in case the system were to crash while everything is being fleshed out. That's one of the reasons why—if you're watching the shutdown—you'll sometimes see, `synching, dirty buffers one, one, one, thirty-seven, twenty-eight, fifteen, two`. You might think—what on earth would cause that? Every process has been killed, why—where do those thirty-eight buffers come from? And it turns out that it's the soft updates code, it's that there are hidden dependencies, and when that one thing writes out, all those dependencies become writable, and all of a sudden you have all these dirty buffers you got to flush out to disk.

On older Unix systems, there was no `reboot` command. There was not a way to even stop the kernel apart from the power switch. If you wanted to `synch` everything out, you'd type `synch`. And you would type `synch` three times. The first one had the effect, and it would schedule all the IO. The other two were an analog delay loop that the programmer was executing to allow time for the stuff to finish before you turned off the machine.

That thinking got instilled in people. And then `reboot` got implemented that was moderately safe for some definitions of safe. The first version I could find flushes the files out, waits five seconds, and says, okay, we're done. Which isn't exactly safe. Usually it's fine, sometimes it's not—you know—if there was a lot to flush out.

So, it's useful to understand how this evolved—why somebody said to type `synch` three times. Maybe you'd say, I'll just put semicolons behind it so that I only have to hit return once, you know—I'll be smart about it. Well, being smart about it doesn't really help because having the individual commands were what made it interesting. And you get all the apocrypha that accumulates around it. Well, the first `synch` schedules it, and the second `synch` blocks until the first `synch` is done—was a common response that I saw in feedback to my blog. And there may be some system somewhere that does that. I didn't examine all systems but I looked at all of the BSD kernels I could find and none of them do that. I looked at Linux kernels and none of them do that. I looked at all the System V kernels that I could find a copy of and couldn't find that in any of those either.

So, it's just a myth that seems to have sprung up on its own because nobody waits for all the current dirty buffers to finish before scheduling the rest. All they do on the `synch` call is literally lock all the buffers, schedule them for IO. Linux will then say to have a barrier and wait for all of them to complete. That was decided a long time ago. Most other systems don't do that. Although there might be some that I'm not aware of that have that as a behavior. There's a lot of diversity in this area because people were trying to make it reliable and a lot of people hacked on it and a lot of solutions came and went in a vacuum and nobody knew what anybody else was doing because all of this was commercial and proprietary.

It's only now that the sources have leaked onto the Internet. You can find these sourc-

es, and you can actually go and study what's going on with the different systems. And one of the difficulties is that some companies were better than others about keeping the source tightly held. For example, you can't find HP sources until it became System V. With earlier ones, you don't know what was going on. If you wanted to study these systems in the original and from the original sources, it could be quite difficult because you would be left with decompiling all this stuff, and that level of effort is beyond my level of interest in the topic.

REUSCHLING: There's a point where you just accept where it is and move on.

LOSH: Exactly. I have the man pages, and what's in the man pages is all I can know.

REUSCHLING: Switching gears a bit, can you tell us a little about Devmatch, what it is, and if you plan to do more work in that area?

LOSH: Yeah, Devmatch is a notion I've had for a long time that I didn't get a chance to implement until more recently. And it started with an observation that we have a bunch of drivers in this system, and every single one of them has these tables that the prog routine will go through and say, does my plug & play information match the first entry? No, does it match the second entry, no, and so on. And I thought, what if the driver could encode a pointer to the table and some metadata about the table's format and put that into the loadable driver? Then you could write a program or leverage an existing program.

Well, if I augmented it to go through and grab this metadata and use that metadata to walk through the tables that were there to create a master table so that when an event comes in you can look at it and go, oh, that event for this kind of device, that driver over there, or that module over there says it has a driver that will accept that device.

For a number of years, Hans Peter had a USB-specific version. He would go through and he would parse, he would put the USB tables in a particular ELF header, and he would parse that out and generate devd rules to load all of the drivers. Starting in FreeBSD 11 that became default. In fact, the period from FreeBSD 10 to FreeBSD 11 is the only time the FreeBSD kernel has shrunk in size. And the only reason that it shrank is that Hans moved all of the USB drivers out of the kernel and put them into devd rules so they would load on demand when you would plug in the different devices.

So, I did that. USB worked, PC card worked, pci kind of worked. It's mostly there now. There may be one or two missing pieces. I haven't done an audit recently. All my work was focused on loading it right after boot. You would have to know to load everything to mount, route, and start, but everything else could be dynamically loaded as part of the boot process.

Manu took that a step further for embedded, and he's run a little bit of code that works with FDT but would be easy to generalize to other things like pci. The hard part was building the code that walked through the binary data structures that are in the loader.hints file that KLDX ref generates. So, we could do something similar for pci, nobody's done that. That's something I would like to find people to work on.

I did a lot of boot loader stuff about the same time I was doing this to get in Lua, and by the time I was done with Lua, I was kind of done with the boot loader for a while. I've not pursued getting this into the boot loader, but that's one way we could make the system even better because then we could load almost every device driver that way. In fact, you can—there's a way you can look at the pci device to know whether or not you need to load it.

And so you could load just a few of them in the boot loader, just enough to get the route filesystem going and go from there. That logic might be a little bit complicated, but it is possible to do. The information exists. And also walking through different binary UEFI data structures is a little bit tedious and time-consuming. Another reason I've not tried to pursue that is you can find the device booting, and you can find the path to it and then find all of the devices along that path and know what to load.

But you know, making that all happen requires more focused effort than I have right now for this project. When I'm doing FreeBSD work, I want to be as focused as possible. So if anybody's looking for a cool project, a cool Summer of Code project—it might be about that size—you can contact me or contact Manu, we can set you up with what you need to do to take the next step in this. So that's what's going on there and what I plan to do about it.

Twelve and thirteen will have about the same level of support unless somebody shows up quickly. I don't think a lot is going to change, but you never know. Somebody else might have their Covid project or turn this into their Covid project.

JUDE: For a lot of devices like network cards and so on, it seems like devmatch makes good sense, and you can decide to load those later and not compile them into the kernel.

LOSH: Yeah, particularly when you have a lot of choices. For storage drivers, there are not so many choices these days. If you have all of the legacy support we had—then there are. And bringing those in on an as-needed basis would also help the kernel size. One of the things I wanted to do with this was to have a more minimal kernel so that we could improve boot time, maybe make it possible to load things on slightly smaller devices, although that hasn't really been a focus for me in a number of years. You know having the ability to do that, having that flexibility to do that, is a good, useful tool that we should finish building because people would use that tool if they had it.

JUDE: Yeah, and just in general, from a security perspective, it's better to have all the code you're not using not loaded.

LOSH: Well, the most secure code is the code that's not on the system.

JUDE: All right, well, thank you very much for taking the time to talk to us. Was there anything else you wanted to talk about before you go?

LOSH: I think I got everything off my chest that I had hoped to get off my chest. I guess one last thing is that if you are interested in 2.11-BSD project, contact me. There are things to do that people can help out with that would make it a better recreation.

Code Review

BY JOHN BALDWIN

Software is written by imperfect people. As a result, software itself is imperfect as it reflects gaps and misunderstandings in developers' perceptions of both the problem being solved and the software serving as the solution. These gaps and misunderstandings manifest as bugs resulting in incorrect behavior, inconsistent behavior, crashes, data loss, etc. One of the tools to aid in mitigating this is having additional developers review source code. While these developers themselves are also imperfect, the various gaps and misunderstandings will vary from person to person. Hopefully, the more developers who are able to examine a given piece of source code, the more likely it is that bugs in the software can be found. In addition, code review can promote "conceptual integrity" as described by Fred Brooks in *The Mythical Man-Month*. In other words, code review by developers familiar with the overall design of a system can help ensure that new changes are consistent with the system's design, or that changes "do things the way this system normally does them."

Over time, the FreeBSD Project has grown a culture of code review. While code review is not mandatory for changes, the ratio of committed changes that are reviewed continues to increase. When I first became involved with the Project in the late 1990s, code review was fairly informal, especially for changes to the source tree. The changes that were reviewed typically took place in private email threads, over IRC, or, on occasion, in person. During code freezes for releases, code review was somewhat mandated in the form of approval by the release engineering team for code freeze commits. However, this approval was generally focused on a risk assessment of the changes relative to the stability of the upcoming release, rather than on the quality of the code itself. Ports developers have had a stronger review culture than the source tree, and several years ago a couple of ports developers stood up an instance of the Phabricator code review tool to review FreeBSD patches. While the initial focus was on providing a better mechanism than Bugzilla for reviewing patches to ports, several source developers began using it as well.

Tools like Phabricator offer several advantages relative to some of the other systems FreeBSD has used. Reviews often include discussions about various trade-offs or assumptions made by new changes. When these reviews occur in private email threads, that knowledge is only available to the individuals on the private thread. When those reviews occur in Phabricator, those discussions are archived and can be found via a URL from the commit that added the changes to the tree. At the same time, Phabricator permits developers to limit which reviews they wish to participate in without requiring all developers to wade through all of the review comments

on every change. Unlike Bugzilla, Phabricator permits comments on individual lines of a patch. Recent versions of Phabricator include the ability for reviewers to type in code snippets as a suggestion that can be applied to a pending change.

Effective code review requires cooperation both from developers submitting patches for review and developers reviewing changes. The rest of this article will focus on best practices when using FreeBSD's Phabricator instance, though similar guidelines are applicable to reviews over other mediums such as email. Some of these guidelines may also be useful in non-FreeBSD contexts.

When submitting patches, a developer should:

- Upload a patch with full context. Uploading patches using the arcanist tool (from the devel/arcanist port or package) will include this by default. If generating a diff from a version control system, force the diff to contain the full context. When using `diff`, `git diff`, or `git show`, add the `-U999999` argument on the command line. To generate a diff in a subversion check-out, use `svn diff --diff-cmd=diff -x -U999999`. Including full context allows developers to see other portions of the code around a change.
- Provide the commit log for the proposed change in the description of the change. This allows reviewers to review both the code and the commit log describing the change. Writing a proper commit log is a topic for another article.
- Tag relevant groups as reviewers. For example, for a change to the PCI bus drivers, add the `#PCI` group. Some groups use rules to auto-add groups for files in certain paths. For example, changes to the bhyve hypervisor are automatically tagged with the `#bhyve` group.
- If the patch submitter has been working with specific developers prior to submitting the patch for review, add those developers as either reviewers or subscribers.
- For FreeBSD source changes, try to format your code based on the guidelines in the `style(9)` manpage. Some of the guidelines can be ambiguous, and patches do not have to be perfect. However, the closer you are to the existing style, the fewer changes you will have to make later on and the more likely other developers will be willing to review your changes and, if necessary, shepherd them into the tree.
- If a reviewer approves your patch but includes comments stating that the approval is conditional on some changes being applied, then the approval is only valid if the patch submitter makes the requested changes. If, on the other hand, a reviewer approves the patch but includes additional comments that are not required for approval, it is up to the patch submitter to decide whether to make the requested changes. Once a patch has been approved, it can be committed without requiring another upload to Phabricator for explicit approval of requested changes.
- Be patient.
- Be responsive to feedback.

Effective code review requires cooperation both from developers submitting patches for review and developers reviewing changes.

When reviewing changes, developers should:

- Review the design first and before smaller fixes like style. If a change is fundamentally incompatible with the system architecture, it is a waste of everyone's time to require a submitter to restyle the patch only to then reject it on architectural grounds. While some general pointers to style rules may make sense when discussing design, the initial review should focus on the design and structure of a patch.
- If critiquing the design or structure of a patch, provide constructive criticism including alternative designs or methods for achieving the desired outcome.
- Respond in a timely manner. Especially if a reviewer has responded and addressed earlier feedback, it is important to approve the change and/or merge it into the tree.
- Be willing to approve a change that still needs some changes if you trust the submitter to make any needed changes before committing. When requesting changes on such a review, be explicit if further changes are either optional suggestions (so can be committed with or without), or if they are mandatory fixes that must be applied before the approval is valid.
- Be willing to recognize that some suggestions are really preferences and categorize them as such rather than as mandatory fixes.
- When noticing a pattern of necessary changes (e.g., a style rule that the submitter might have misunderstood or not known), rather than pointing out every flaw on the first review, point out a single instance with a reference to the general rule and let the submitter have a chance to resolve the issue throughout the patch.

JOHN BALDWIN is a systems software developer. He has directly committed changes to the FreeBSD operating system for 20 years across various parts of the kernel (including x86 platform support, SMP, various device drivers, and the virtual memory subsystem) and userspace programs. In addition to writing code, John has served on the FreeBSD core and release engineering teams. He has also contributed to the GDB debugger and LLVM. John lives in Concord, California, with his wife, Kimberly, and three children: Janelle, Evan, and Bella.

Thank you!

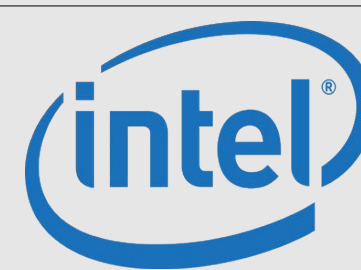
The FreeBSD Foundation would like to acknowledge the following companies for their continued support of the Project. Because of generous donations such as these we are able to continue moving the Project forward.



Are you a fan of FreeBSD? Help us give back to the Project and donate today! freebsd.foundation.org/donate/

Please check out the full list of generous community investors at freebsd.foundation.org/donors/

Uranium



Iridium



Platinum

NETFLIX

Gold



Silver

BECKHOFF



STORMSHIELD



Tarsnap

vmware



Writing Good FreeBSD Commit Messages

BY ED MASTE

Why are commit messages important?

When you commit a change in Git, Subversion, or another version control system (VCS), you're prompted to write some text describing the commit—a commit message. How important is this commit message? Should you spend some significant effort writing it? Does it really matter if you write simply `fixed a bug`?

Most projects have more than one developer and last for some length of time. Commit messages are a very important method of communicating with other developers in the present and for the future.

FreeBSD has hundreds of active developers and hundreds of thousands of commits spanning decades of history. Over that time the developer community has learned how valuable good commit messages are; sometimes these are hard-learned lessons.

Commit messages serve at least three purposes:

- **Communicating with other developers**

FreeBSD commits generate email to various mailing lists. These include the commit message along with a copy of the patch itself. Commit messages are also viewed through commands like `git log`. These serve to make other developers aware of changes that are ongoing; that other developer may want to test the change, may have an interest in the topic and will want to review in more detail, or may have their own projects underway that would benefit from interaction.

- **Making changes discoverable**

In a large project with a long history it may be difficult to find changes of interest when investigating an issue or change in behavior. Verbose, detailed commit messages allow searches for changes that might be relevant. For example, `git log --since 1 year --grep 'USB timeout'`.

Commit messages are a very important method of communicating with other developers in the present and for the future

- **Providing historical documentation**

Commit messages serve to document changes for future developers, perhaps years or decades later. This future developer may even be you, the original author. A change that seems obvious today may be decidedly not so much later on.

The `git blame` command annotates each line of a source file with the change (hash and subject line) that brought it in.

Having established the importance, here are elements of a good FreeBSD commit message:

- **Start with a subject line**

Commit messages should start with a single-line subject that briefly summarizes the change. The subject should, by itself, allow the reader to quickly determine if the change is of interest or not.

- **Keep subject lines short**

The subject line should be as short as possible while still retaining the required information. This is to make browsing `git log` more efficient, and so that `git log --oneline` can display the short hash and subject on a single 80-column line. A good rule of thumb is to stay below 63 characters and aim for about 50 or fewer if possible.

- **Prefix the subject line with a component, if applicable**

If the change relates to a specific component the subject line may be prefixed with that component name and a colon (:).

✓ `foo: Add -k option to keep temporary data`

Include the prefix in the 63-character limit suggested above so that `git log --oneline` avoids wrapping.

- **Capitalize the first letter of the subject**

Capitalize the first letter of the subject itself. The prefix, if any, is not capitalized unless necessary (e.g., `USB:`).

- **Do not end the subject line with punctuation**

Do not end with a period or other punctuation. In this regard the subject line is like a newspaper headline, or the subject header in an email message.

- **Separate the subject and body with a blank line**

Separate the body from the subject with a blank line.

Some trivial commits do not require a body and will have only a subject.

✓ `ls: Fix typo in usage text`

- **Limit messages to 72 columns**

`git log` and `git format-patch` indent the commit message by four spaces. Wrapping at 72 columns provides a matching margin on the right edge. Limiting messages to 72 characters also keeps the commit message in formatted patches below RFC 2822's sug-

Commit messages
are a very important
method of
communicating
with other
developers in the
present and for
the future

gested email line length limit of 78 characters. This limit works well with a variety of tools that may render commit messages; line wrapping might be inconsistent with longer line length.

- **Use the present tense, imperative mood**

This facilitates short subject lines and provides consistency, including with automatically generated commit messages (e.g., as generated by `git revert`). This is important when reading a list of commit subjects. Think of the subject as finishing the sentence “when applied, this change will ...”.

- ✓ `foo: Implement the -k (keep) option`
- ✗ `foo: Implemented the -k option`
- ✗ `This change implements the -k option in foo`
- ✗ `-k option added`

- **Focus on what and why, not how**

Explain what the change accomplishes and why it is being done, rather than how.

Do not assume that the reader is familiar with the issue. Explain the background and motivation for the change. Include benchmark data if you have it.

If there are limitations or incomplete aspects of the change, describe them in the commit message.

- **Consider whether parts of the commit message could be code comments instead**

Sometimes while writing a commit message you may find yourself writing a sentence or two explaining some tricky or confusing aspect of the change. When this happens consider whether it would be valuable to have that explanation as a comment in the code itself.

- **Write commit messages for your future self**

While writing the commit message for a change you have all of the context in mind—what prompted the change, alternate approaches that were considered and rejected, limitations of the change, and so on. Imagine yourself revisiting the change a year or two in the future and write the commit message in a way that would provide that necessary context.

- **Commit messages should stand alone**

You may include references to mailing list postings, benchmark result web sites, or code review links. However, the commit message should contain all of the relevant information in case these references are no longer available in the future.

Similarly, a commit may refer to a previous commit, for example in the case of a bug fix or revert. In addition to the commit identifier (revision or hash), include the subject line from the referenced commit (or another suitable brief reference). With each VCS migration

Do not assume that the reader is familiar with the issue. Explain the background and motivation for the change. Include benchmark data if you have it.

(from CVS to Subversion to Git) revision identifiers from previous systems may become difficult to follow.

- **Include appropriate metadata in a footer**

Commit messages may have one or more of a number of standard metadata tags in the final paragraph. Standard tags used in FreeBSD are:

Tag	Description
PR	FreeBSD problem report (Bugzilla) number
Submitted by	ID the original author, if not the committer
Reported by	ID of a third party who reported the issue
Reviewed by	Reviewer ID
Tested by	ID of those who have tested the change
Approved by	Mentor or code owner who approved the change
Obtained from	Source of a change in another project
MFC after	Time period before merging the change from Current to Stable
MFC with	Associated commit that this change should be merged along with
MFH	Ports quarterly branch for merge request
Relnotes	Yes/No whether this change should be included in release notes
Security	External reference for a security issue, such as a CVE number
Sponsored by	Organization or event that sponsored work on the change
Differential Revision	Full URL of code review in FreeBSD’s Phabricator instance

“ID” indicates either a FreeBSD userid or a name and email address. Multiple IDs may be presented as a comma-separated list or by repeating metadata tags on subsequent lines.

ED MASTE manages project development for the FreeBSD Foundation. He is also a member of the elected FreeBSD Core Team. Aside from FreeBSD, he has contributed to a number of other open-source projects, including LLVM, ELF Tool Chain, QEMU, and Open vSwitch. He lives in Kitchener-Waterloo, Canada, with his wife, Anna, and sons, Pieter and Daniel.

How to Contribute Without Being a Programmer?

Become a FreeBSD Translator!

BY DANILO G. BAIO



The FreeBSD Project is adopting Weblate as its web-based, continuous, localization platform, which makes it easy for people to become translators. It began operation at the start of this year and so far has gathered 10 languages and 89 contributors to help translate the FreeBSD documents.

Language	Code
Chinese (Simplified)	zh_CN
Chinese (Traditional)	zh_TW
French	fr_FR
German	de_DE
Italian	it_IT
Norwegian	nb_NO
Persian	fa_IR
Portuguese	pt_BR
Spanish	es_ES
Turkish	tr_TR

The list shows only the current languages in operation on our Weblate instance—new languages are very welcome. This list doesn't show the current translations that are already committed in the FreeBSD Doc repository.

With Weblate, it's so easy to translate that contributors don't need to handle a repository, an xml file, builds, or to understand DocBook/XHTML. They just need to learn some cool things and translate it to their native language in the process.

Focused on learning, the FreeBSD Documentation is very known for its quality. When you begin to translate any document, I can assure you that you will understand why.

The full set of documents contains 35 articles and 8 books. Here are the current statistics from Weblate (<https://translate-dev.freebsd.org/projects/freebsd-doc/#information>).

	Strings	Words
All documents	39,779	691,513

The *FreeBSD Handbook* is a large document and currently the main document of the Project.

	Strings/sentences	Words
FreeBSD Handbook	12,212	254,554

There are smaller documents as well—articles for example:

	Strings/sentences	Words
Contributing to FreeBSD	217	5,622

They all can be accessed on <https://www.freebsd.org/docs/books.html>.

For new languages, we recommend starting with one or two small documents before heading to the *FreeBSD Handbook* so that people can get used with the tool and also to see their work committed faster.

When translating you need to give your full attention to the text, understand the context, and occasionally re-read it, but that’s a good thing and it’s how you will learn in the process. I still remember some specific details of things I translated in the past.

Simple steps to start contributing:

- **Create an account, with an email address or with your Github id**

<https://translate-dev.freebsd.org/>

Weblate creates new commits for each translation and references them with your email address, so it’s easy for you to track your work. Your translations will appear on your Github Contribution Activity so long as you use an email address that matches with your Github account.

- **Subscribe to the freebsd-translators mailing list**

<https://lists.freebsd.org/mailman/listinfo/freebsd-translators>

Updates in the infrastructure and all news about translations are sent there. If you have any doubt, that’s the place for you to check.

- **Introduce yourself in the mailing list and ask to join a language team**

If the language team does not exist, ask to create it—you could be the coordinator.

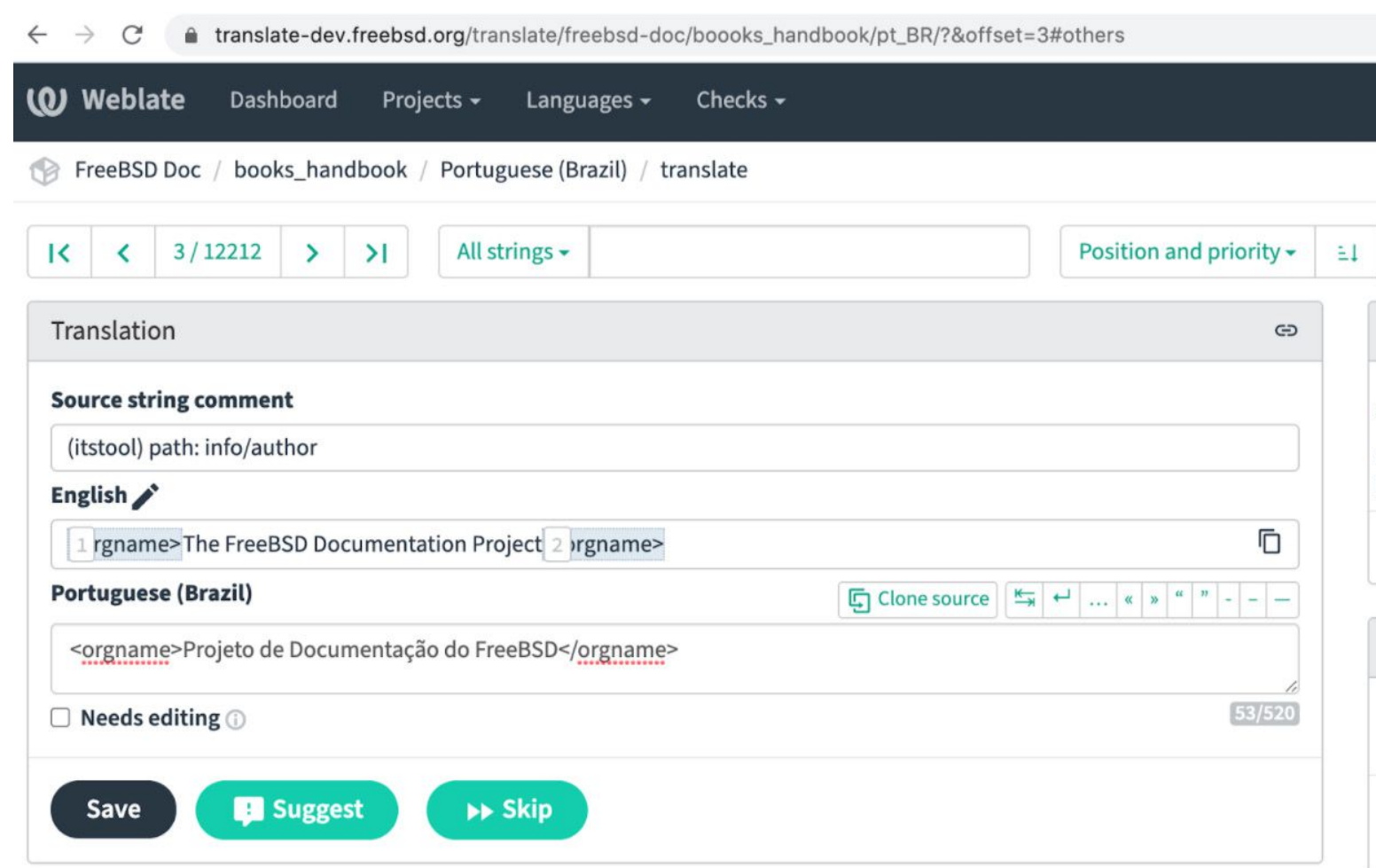
The introduction is important because you can discover other people translating into the same language and make it easier to coordinate efforts.

- **Log in to Weblate with your new account**

After your introduction, an admin or coordinator will add your username to a language team and you can start translating.

- **Find your language team and choose an initial document to translate**

Translating online on the Weblate interface is the easiest way to contribute, once a coordinator or admin has given you access to a language. The save button will be enabled and you can start working.

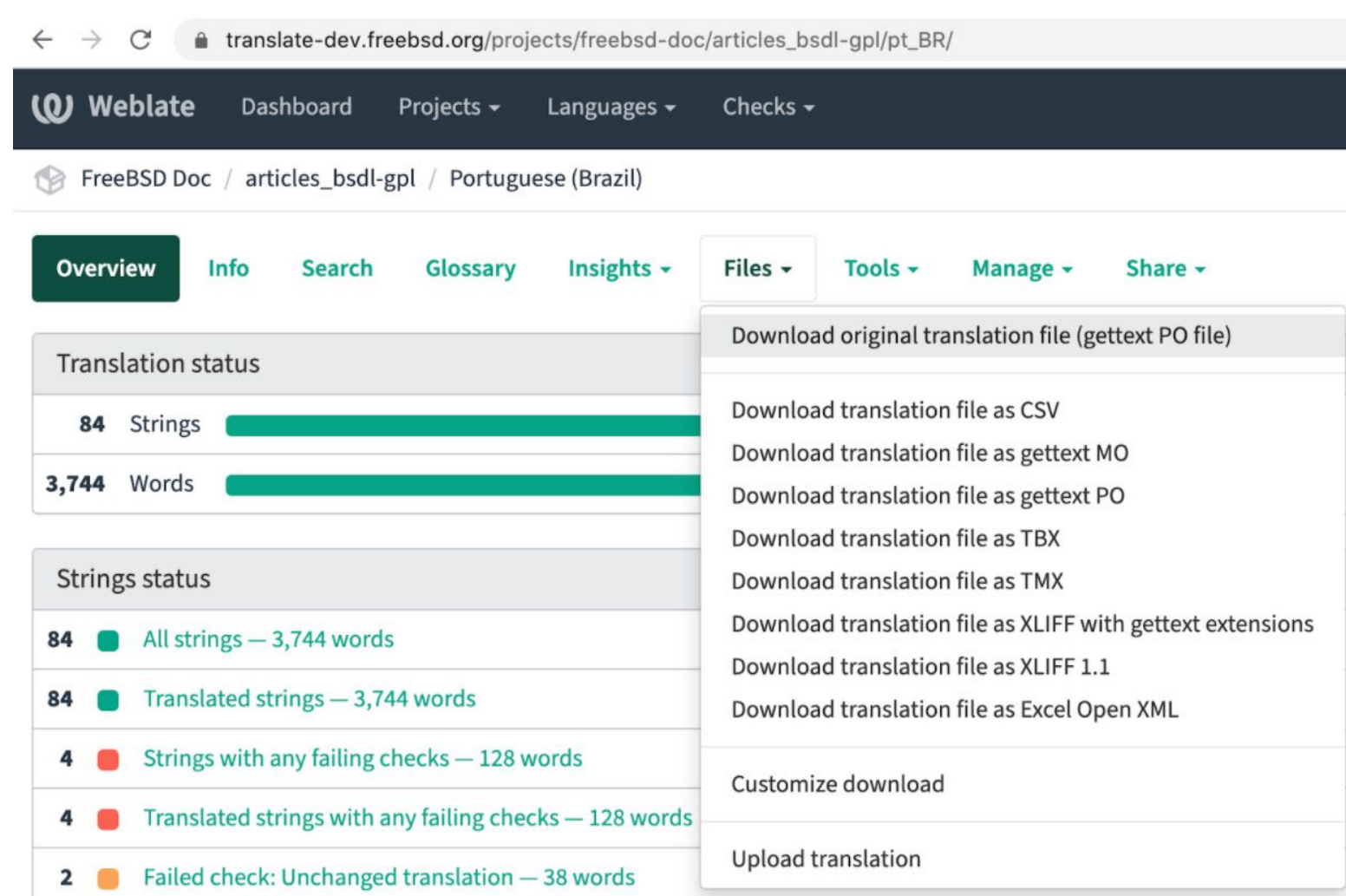


Weblate has a set of links that lead to actual translation. The translation is further divided into individual checks, like *Untranslated* or *Needing review*. If the whole document is translated, without error, the *All translations* link is still available. Alternatively, you can use the search field to find a specific string or term. You can find out more info about translations in the official Weblate documentation—things like keyboard shortcuts and other tips about the translation tool, but the interface is very intuitive.

<https://docs.weblate.org/en/latest/user/translating.html#translation-projects>

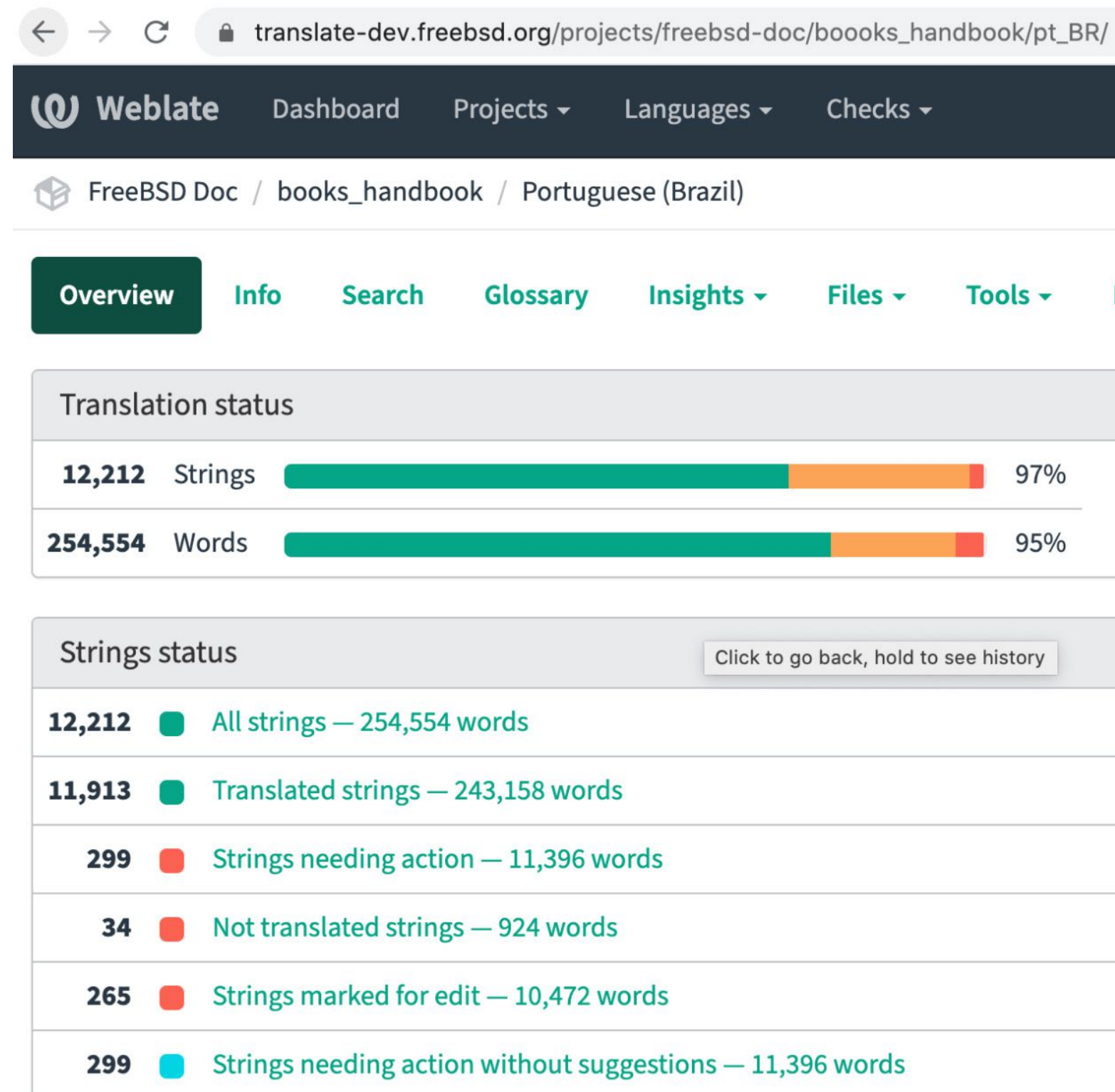
Translating offline

If you are familiar with PO Gettext and would like to make offline translations, you can download and upload your translations through the document page of your language by clicking in the Files section.



Proofreading and Weblate Quality Checks

When you click on *Project / Language / Document*, the Translation and String Status from Weblate for that document will be shown. This page is very useful for proofreading and quality checks.

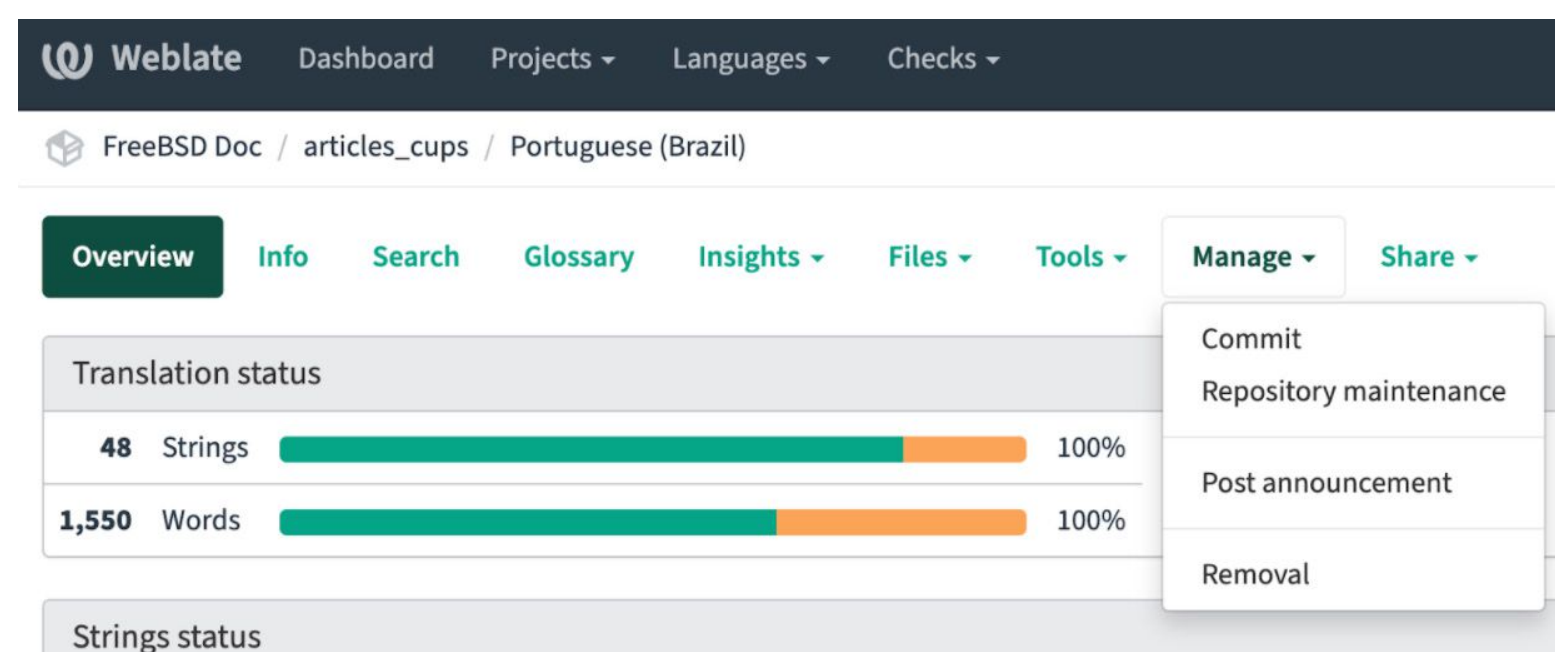


In this example, there are some strings needing revision. If you click on these links, it will show only those strings that need to be revised/translated.

It is often more helpful to see the translated strings in their final context in order to do the proofreading more easily, and our project has some facilities for this.

Building the Translated Document

All translations in our Weblate instance are built daily and they are available on <https://doc.fugbr.org/jenkins/>. We enable these builds for all new languages. If you want to speed up this process, you can tell Weblate to commit your translations any time you want and then a build process will be triggered immediately through our Jenkins, and some minutes later you can proofread it.



You can also build the documents locally. There is more information about that on our wiki page on <https://wiki.freebsd.org/DocTranslationOnWeblate>.

Submitting Translations

After finishing any document translation, submittal will be the last step to push your work officially to the FreeBSD website.

In the book *FreeBSD Documentation Project Primer for New Contributors*, there is a chapter explaining how to submit translations. It is not difficult and if this is your first document, we will have people help you in the process. https://www.freebsd.org/doc/en_US.ISO8859-1/books/fdp-primer/po-translations-submitting.html

After committing translations to the official documentation tree, all translators' names will be added to the list of *Additional FreeBSD Contributors*. Congratulations! Now you are part of the FreeBSD history as well.

Final Thoughts

For translating, you don't need to be a developer, you just need to be willing to contribute, and as noted before, you will learn some new things in the process.

There are developers who started contributing to FreeBSD Documentation and later become ports or kernel developers, this can be an initial journey for you in the Project.

In the FreeBSD Documentation, subjects are broader and there are pieces of Unix and BSD history there, stories about amazing contributors who sadly aren't with us anymore. For instance, read the Bruce D. Evans In Memoriam text—it's in the article *Contributors to FreeBSD*, chapter *Development Team: In Memoriam*, which I only became aware of when translating it.

It's really nice to know the history and be part of it as well. To me, being part of the FreeBSD Project is such an honor.

If you have any questions, join us on `freebsd-translators@` mailing list. You will be very welcome. Please help promote this effort to your local user group as we always need more volunteers.

DANILO G. BAIO is a FreeBSD ports committer and Brazilian Portuguese translator.
dbaio@FreeBSD.org

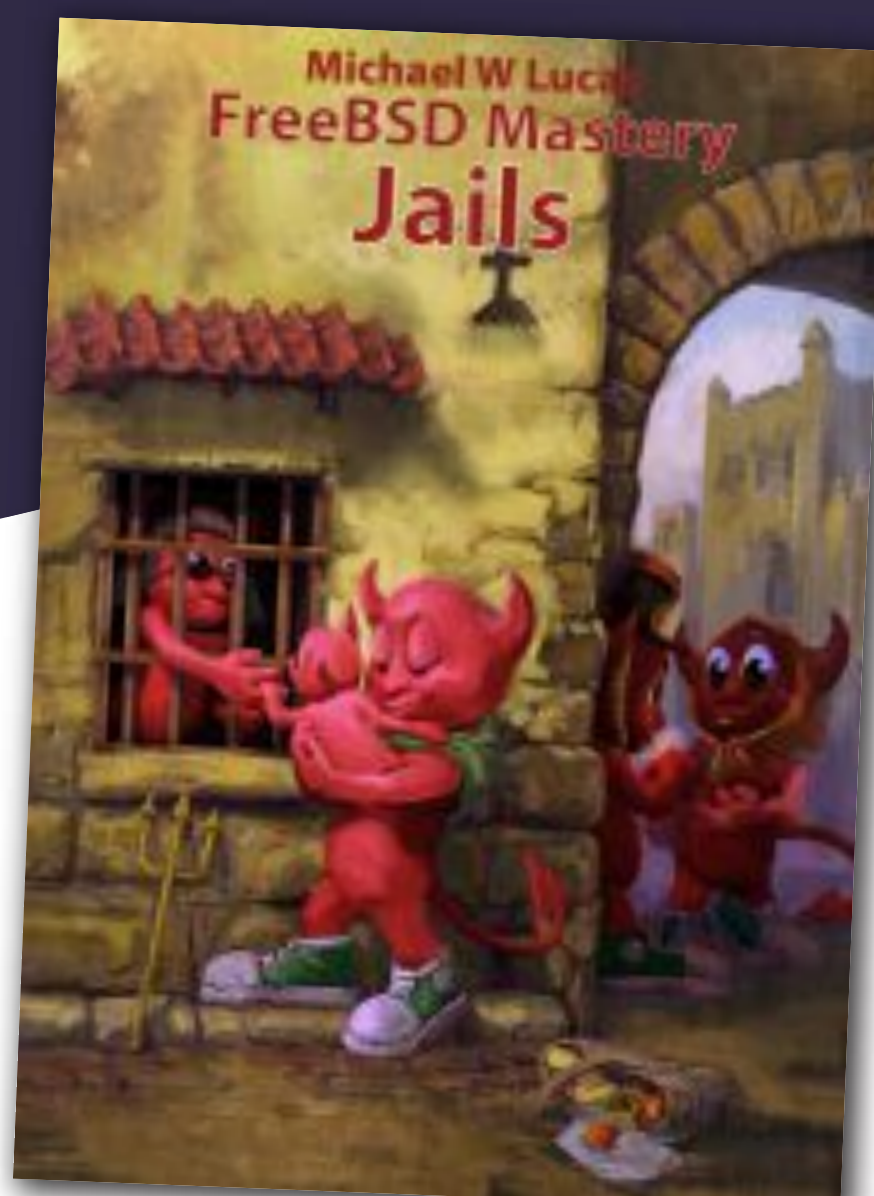
Jails ARE FBSD'S MOST LEGENDARY FEATURE:

KNOWN TO BE POWERFUL, TRICKY TO MASTER,
AND CLOAKED IN DECADES OF DUBIOUS LORE.

FreeBSD Mastery: Jails cuts through the clutter to expose the inner mechanisms of jails and unleash their power in your service.

Confine Your Software!

- * Understand how jails achieve lightweight virtualization
 - * Understand the base system's jail tools and the iocage toolkit
 - * Optimally configure hardware
 - * Manage jails from the host and from within the jail
 - * Optimize disk space usage to support thousands of jails
 - * Comfortably work within the limits of jails
 - * Implement fine-grained control of jail features
 - * Build virtual networks
 - * Deploy hierarchical jails
 - * Constrain jail resource usage.
- ...**And much, much more!**



FreeBSD Mastery: Jails BY MICHAEL W LUCAS **Available at All Bookstores**



FreeBSD[®] JOURNAL

The FreeBSD Journal is Now Free!

Yep, that's right Free.

The voice of the FreeBSD Community and the BEST way to keep up with the latest releases and new developments in FreeBSD is now openly available to everyone.

DON'T MISS A SINGLE ISSUE!

2020 Editorial Calendar

- FreeBSD in Research (Jan/Feb 2020)
- Filesystems (March/April 2020)
- Network Performance (May/June 2020)
- Benchmarking/Tuning (July/August 2020)
- Contributing/Onboarding (Sept/Oct 2020)
- Workflows (Nov/Dec 2020)



Find out more at: freebsd.foundation/journal



How to Become a Doc Committer

BY BENEDICT REUSCHLING



The FreeBSD documentation set is comprised of various handbooks for users and developers, FAQs, articles on specific topics, man pages in the src repository, and the main FreeBSD.org website. It is a great way for newcomers to make their first contribution to the FreeBSD Project and has been the gateway to many developers still active. While reading our documentation, people often find things that need correction: from a missing comma, to a typo in a word, to a whole paragraph that is outdated and needs rewriting. Those who do not shrug their shoulders and move on either report the bug on our Bugzilla [https://bugs.freebsd.org/bugzilla/enter_bug.cgi?product=Documentation] system or come up with a patch to fix the issue. While the latter is certainly not a requirement (especially for drive-by reports and people who do not yet know our doc toolchain), the former is still better than doing nothing about it.

It's also not just about finding bugs in existing material. If you have found an easier way to do something and can explain it, submit a bug report with your work. Other people may find that our handbook could do with more pictures to illustrate the concepts described there (after all, a picture is worth a thousand words) and may want to send us their sketches. Updated screenshots also fall into this category and help newcomers figure out that they are on the right track by comparing them with what's on their screen. Flyers and cheat sheets with common FreeBSD commands to hand out at conferences are another way to show your talent. Did I mention that we could use more EXAMPLE sections in various user man pages? Run link checkers on a regular basis on web pages we link to and report dead links (even better, find an alternative location). If you like wikis and make edits there, join the #freebsd-wiki group on the Freenode IRC. In other words, surprise us (in a good way) and show up with some examples to demonstrate that you are serious about your endeavor.

Whatever it is, a documentation committer will pick up your work (typically a patch or a textual description) and start working on it. If not, a gentle nudge may help, as developers are dealing with a lot of work. In the simplest case (comma or typo fix), they'll make changes to the sources directly and commit it, giving you as the original submitter the credit in the "Submitted by:" line of the commit message. Thank you for your contribution, you just made the FreeBSD documentation a little better!

In other instances (when it gets a bit more complicated or the patch is bigger), you can also create an account on our Phabricator instance [<https://reviews.freebsd.org/>] and submit the patch that way (adding the docs group or individuals as reviewers). This makes the patch review process easier to track. It allows feedback from various angles (writing style, formatting, etc.) and updates of an existing patch. Don't get frustrated if you only get a few lines of feedback that may sound

overly critical. In most cases, I've found that people who do reviews have only a few minutes to do these and thus keep their feedback short and simple. Don't be afraid to ask follow-up questions if you don't understand any of the jargon we use or don't know how to proceed. Don't take feedback as personal as it is not you we criticize—just the code. I typically try to mix some positive feedback into my comments. "Thank you for the patch," "Good catch," and "Glad you took the time to report this" can go a long way toward making a submitter feel good and willing to work the provided feedback into their code for another review round.

With time, patience, and some practice, you'll get your first contribution into the documentation tree. It feels good, right? You'll probably show this to your friends and colleagues, post it on social media, shout it from the highest mountain. I think people want these successes in their lives where they can point at something and proudly say "there, that's what I made." While some may say: "Huh, big deal," remember that it is based on your contribution that a committer took up this patch and made a change in the FreeBSD repository. You've become part of project history, a project whose roots go back over 30 years and counting. It's also a great way to work with people on something you care about. Many people like this so much that they continue to contribute other patches. And that's where the trouble begins...

Over time, the same person may submit a lot of good additions to the documentation tree. From a flood of small patches over a couple of weeks to groundbreaking, new work adding much more material to our documentation (and anything in between). People in the project start noticing. In addition, you take feedback to heart and work with developers in a constructive and collaborative way. Then one day it happens: the committer gets annoyed, because they are doing all this work for you that you could basically do yourself. That is what we say when we "punish" someone with a commit bit.

The doc committer will create a list of your past deeds (some may call it a special form of a "naughty list"). That is why we collect your track record in the "Submitted by:" field of the commit message. Phabricator can also generate a nice list of all your review interactions. This list is then sent to the Documentation Engineering Team (abbrv. doceng), which is (among other things) responsible for managing the documentation commit bits. A usual plea of "please let me mentor that person so that I can get my time back while they commit their own work" goes along with it. This proposal is then evaluated and voted on internally. Sometimes committers form a small tag team to better cover different time zones between mentor and mentee (faster feedback loops). The doceng team can pick additional mentors if they deem it necessary. If the vote comes out in favor of the mentee, they are given notice and granted access to Project resources necessary to work with the documentation. If the commit bit is not granted, we usually provide feedback on things that are missing (usually "beef up your track record") and encourage another proposal again after some time has passed.

The mentor will then teach the new mentee everything they need to know: from understanding our formatting rules as well as bumping the .Dd (document date) of a man page when there is a significant enough change being made, to their first nervous (often sweaty-handed) commit. During this time, the mentor will help the mentee when they make a mistake. The mentoring phase is meant to give the mentee enough confidence to not make these mistakes again and learn from the vast experience the mentor has from working with the Project. Social interactions from the mentee within the community are also a part of the process as well as welcoming feed-

With time, patience, and some practice, you'll get your first contribution into the documentation tree.

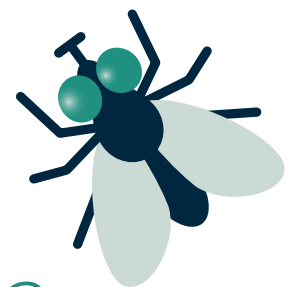
back from contributors (which completes the circle), working with other members of the community, and oftentimes committing their own submitted patches after mentor review.

Modern tooling like Phabricator makes this a much easier process than doing it over email, which works (that's how I was mentored way back when), but it takes longer. It's some intense work, but in my experience, also a lot of fun. I've seen mentors and mentees form a deep personal relationship from the process, which is a satisfying experience in itself. Each mentee is different, of course, but more often than not, I've seen a mentor and mentee meet at a conference for the first time and immediately talk like they've been old friends.

Some people in the project don't think it is worthwhile to mentor because it takes too much time away from their own FreeBSD work. That may be true, although I've found that the reduction in my work is paid back multiple times in added productivity for the Project as a whole. After all, two people are making changes and the passion and enthusiasm of a new mentee might motivate you to take on that long-dreaded documentation change together. Mentoring is not for everyone and so no one is forced to mentor. Luckily, there are enough people willing to do that and some are willing to co-mentor someone to share the load. In my experience, it's a tribe that mentors someone and not a single person. Feedback and encouragement are given not just by the mentor alone, which encourages the mentee to cast a wider net over time. I've seen many former mentees of mine move into other areas of the Project where I wouldn't go because my knowledge in that space is limited. But what a great benefit for the Project as a whole.

Be that as it may, the mentoring period may take a while, depending on the amount of work available, time constraints, and engagement of both mentee and mentor. After a while, mentors feel that they just approve changes without the need to do much correcting anymore and that the mentee has learned all there is to know. With the agreement of the mentee, they are released and are then an equal documentation committer like all the others. I personally encourage my former mentees in my passing message to continue to ask me questions if they are unsure. Also, I tell them the wise words of Yoda to "pass on what you have learned," meaning that they should take their own mentee when the time comes. I have often mentored other people with former mentees of mine, and have seen that they put the same kind of effort into the mentoring process.

Did I get you interested? Take a look at the FreeBSD Documentation Project Primer for New Contributors [https://www.freebsd.org/doc/en_US.ISO8859-1/books/fdp-primer/]. It shows you how to check out the documentation tree and how to make some changes to existing docs. The Quick Start section has these things covered. If you are interested in man-page changes, it has its own section. Other parts deal with formatting, formatting elements, and how to properly configure your editor to make your life a little easier. It's not too difficult to get started and who knows where this will lead to over time?



*I tell them the
wise words of Yoda
to "pass on what
you have learned."*

BENEDICT REUSCHLING is a documentation committer in the FreeBSD project and member of the documentation engineering team. He serves on the board of directors of the FreeBSD Foundation as vice president. In the past, he served on the FreeBSD core team for two terms. He administers a big data cluster at the University of Applied Sciences, Darmstadt, Germany. He's also teaching a course "Unix for Developers" for undergraduates. Together with Allan Jude, he is host of the weekly bsdnow.tv podcast.

Google Summer of Code

BY MARIUSZ ZABORSKI

Google Summer of Code (GSoC) 2020 ended in September. This amazing initiative allows young students to gain a lot of experience while contributing to open-source organizations. FreeBSD project have participated in the GSoC each year from the start. In this article, we introduce this project from the student side and the mentor side.

Google Summer of Code (GSoC) is an annual, international program that Google started in 2005. The goal is to expose students to an open-source project, and the program is designed as a vacation scholarship for students. Instead of looking for internships or going on vacation, students can spend their summer coding for an open-source organization.

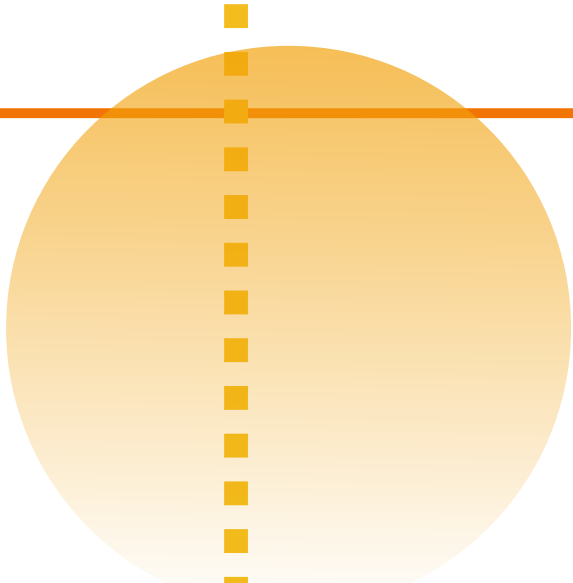
Students are obligated to work almost full time on the project of their choice, and Google funds a stipend for them. The amount varies each year. Currently, the scholarship amount depends on the student's location and it ranges from \$3,000 to \$7,000 USD.

Fortunately, the students are not alone, as GSoC requires students to work with a mentor. The mentors are members of, or contributors to, an open-source organization that is interested in the topic of the student's work.

GSoC starts way before summer. Google opens applications in January of each year and during this time, open-source projects can apply to join the program. A good practice is to prepare even earlier and constantly collect ideas for GSoC projects. Create a simple wiki page or Google Doc and collect potential projects for students to work on. The projects can vary from something simple to something very complicated, but you have to keep in mind that students should be able to provide some kind of code during the summer. It is very important to offer interesting projects; otherwise, students may choose a different open-source organization. Also, remember that a person from the organization (the mentor) should be responsible for the project. This person has to be able to help the student on their journey.

After a couple months, the open-source organizations are granted or denied acceptance in the program—they usually received word at the end of February. After that, students are given some time to look for open-source organizations and projects they would be interested in. They can discuss the ideas through mailing lists, IRC, Slack, or anywhere else. If they aren't interested in any of the projects listed, students can suggest a project themselves and then an open-source organization can look for a mentor for the project.

In March, the open-source community reviews the students' applications and assigns mentors. GSoC organizers inform each open-source project of how many seats they have. Unfortunately, there won't be enough seats for all students. The applications will be reviewed based



on the quality of the research done by the student, their knowledge of the subject, and their level of engagement in the last few months. Some good advice for students is to show interest as early as you can, think about the challenges of the project, do some research, discuss the challenges with the community, and even look into the code a little. Nobody expects you to know the code like the

back of your hand—your mentor will probably help you with the source code—but going the extra mile to look into things yourself demonstrates your engagement.

Mentors should not only have expertise in the project's subject matter but should also be integrated with the open-source community. If there is a problem they can't help with, or the student needs additional access to something, the mentor should know where to go. They should know the project code guide and be able to commit the code or get someone to commit the code. For some students, getting their first commit to open source may be more important than a scholarship (and you want to focus on those). Some projects may have more than one mentor.

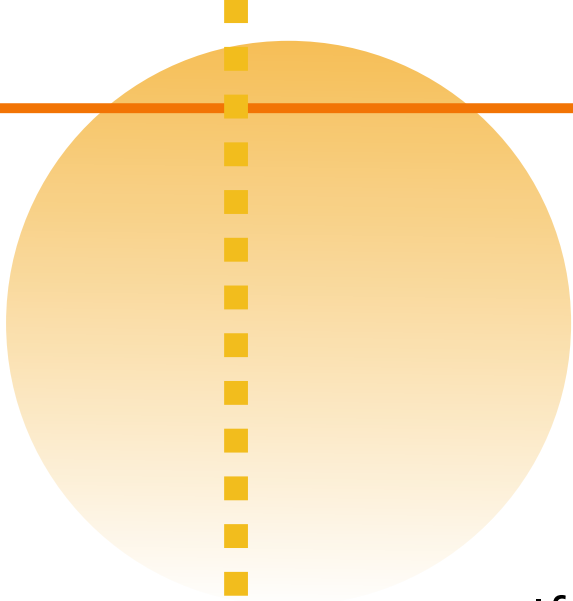
After students are accepted, the community bonding and coding starts. This is the time for students' hard work. At FreeBSD, we have a guide for what students should do during this period. One of the things is to maintain a wiki page of their project. Students should try to update it as often as they can, which helps with project visibility in the organization. It is good to document the issues and challenges that we encounter during the summer because many projects won't be finished by the end of the program. By documenting the work that was done, we can pass it on to somebody else after GSoC.

Students should also send weekly status updates to the mailing list. This activity motivates students to share and document what they have accomplished and helps with the project's visibility. It also helps with community bonding. There may be some developers in the group who can help with an issue or have a different view of the project. It is really worth sharing status updates.

Last but not least, students should push their code as often as possible. The code quality doesn't really matter—you can always improve it—but it is much, much easier to discuss something when you see the code. Some students send code only once or twice a month. That doesn't work. After a month of coding, we may find that we should go in a totally different direction, and it is hard for mentors to help students when they don't see the code. Mentors should encourage students to push their code often and not be afraid to show work in progress.

Most of the time, the project is done online. Only a few students have the chance to work with their mentors on-site every day. Communication may be hard sometimes, especially if the mentor and students are in different time zones. Emails between students and mentors may seem too formal and students may be reluctant to write emails. Many students may also treat the relationship like a professor-student relationship. If you never see your mentor in person, it can be hard to imagine how they will react. Students and mentors should try to reduce the communication overhead. Communicating over an IRC, Slack, or Discord is much better for this purpose than email.

Mentors should also remember that the project is theirs, and they are responsible for it. If you don't respond to questions for weeks, then the student may be discouraged from working on the project. You have to be there for the students and help them, but remember that the real work should be done by the students. You're just there to help manage the process and help with communication with the community that you know so well.



Mentors evaluate their students a few times over the summer. (In previous years, there were two evaluations. Now, there are three.) The mentors report the students' progress to Google. Remember, if a mentor and student don't submit the evaluation on time, Google will cancel the project. Google pays the students after each successful evaluation.

If the project is going well, then great and it's not complicated. The mentor simply passes the student on to the next stage. If the student hasn't written a single line of code, then the next part is very unpleasant but still simple.

Unfortunately, there is also a middle ground. Some students are very active at the beginning of the project and then suddenly disappear. Some students rush and do all the work just before the evolution. Sometimes, it is hard to know how much work the student has done because of the remote working situation (this is why code pushing is so important). The most important thing is for mentors to ask themselves if they still want to work with the student. If not, they have to make a hard decision.

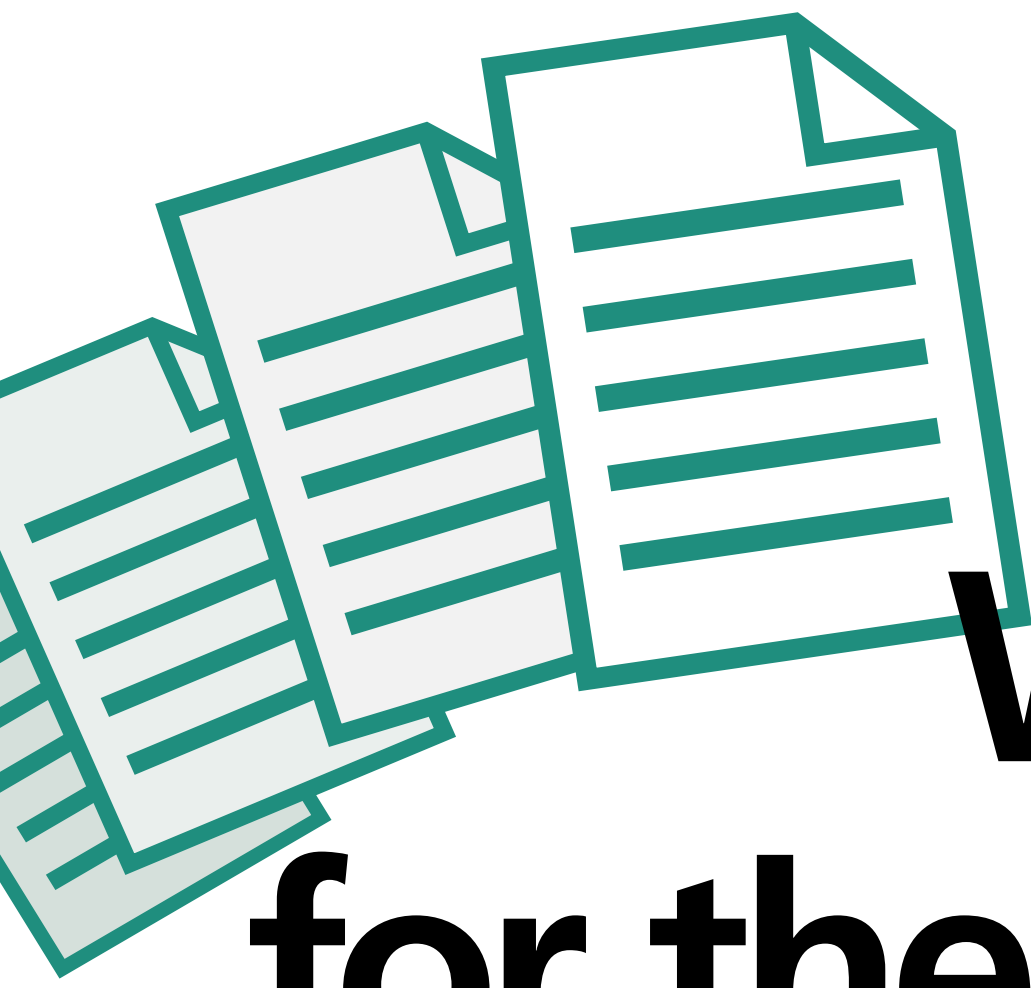
If there is still hope, mentors must tell their students what they should improve—and if they don't perform better, then they will fail them on the next evaluation. If you are doing more work than your student, then you have to fail the student. If you constantly struggle to engage your students, then you have to fail them. Student engagement is much more important than the quality of the code they produce.

There is no better way for open-source organizations to obtain fresh blood. Most students are not exposed to niche open-source projects. GSoC allows students to get to know a project from the inside out. They may contribute to the project (if they do, that's perfect), but it's not the most important part. The most important thing is to show them a great open-source environment and allow them to learn from it. If we do that, more and more students will stay in this environment, and everyone will gain from it. The students will gain knowledge that would otherwise be hard to get and earn nice CV entries that may open some new doors for them.

History shows that a lot of students who started a GSoC project continued to contribute to the project many years later. In this author's opinion, that should be the goal of any open-source organization involved in GSoC. We have to remember that the students' code won't be ideal, but if they stay on the project for longer than those three months, the gains for the organization will be much more significant than just this single project. Many students go on to become GSoC mentors the next year!

We hope to see you as a student or mentor at the next GSoC!

MARIUSZ ZABORSKI is a QA and Dev manager at Fudo Security. He completed a successful Google Summer of Code project in 2013 and mentored a few GSoC students for the FreeBSD Project. Mariusz has been the proud owner of his FreeBSD commit bit since 2015. His main areas of interest are OS security and low-level programming. At Fudo Security, Mariusz leads a team developing the most advanced solutions for monitoring, recording, and controlling traffic in IT infrastructure. In 2018, Mariusz organized the Polish BSD User Group. In his free time, he enjoys blogging at <https://oshogbo.vexillum.org>.



Writing an Article for the *FreeBSD Journal*

BY ROLLER ANGEL

I'm sure you've learned many cool things about FreeBSD along the way, just as I have. And some of those things you probably wish you'd known earlier than you did, just as I did. Think about the thousands of users out there, on all kinds of levels, and the many paths there are to learning more about FreeBSD—including reading. There are manuals and guides and books. There are also articles as we have here in the *FreeBSD Journal*. Someone has to write these articles, and I'm here to tell you that writing is an interesting, fun way to learn more about FreeBSD too. I've learned more than I ever would have imagined when I started out on this writing path.

But who writes about FreeBSD? You may be thinking someone who has been in the field for decades or someone of a certain experiential level. The truth, however, is that it could be you. Did you ever learn something—a new process or way of approaching a problem, an avenue you hadn't considered, something that simplified how to get from Step A to Step B—from talking to a friend or coworker? What was it about that friend's vision or explanation or understanding that made your understanding clearer or deeper? What was it about yours that helped your friend?

While writing a FreeBSD-themed article may seem like a challenge at first, it is really just like having a discussion with a colleague about a topic that interests both of you, or talking with friends about how you implemented something on your computer. Of course, to write an article for a broader readership, you'll likely need to do a little research and maybe refine your writing skills to get your point across in a way that your reader will learn something new or understand how someone else uses software they also use. You'll want to think about who will read your article, why they're reading it, what they bring to the process, and what you want them to learn from your experience, perspective, and/or explanation.

Think about something you've done recently that someone in the FreeBSD community would be interested in hearing about or could learn from. Sharing your personal experience and expertise in that way will probably be more valuable to others than you might think. Your topic might be something you just discovered and wish someone had told you about years ago, or, on the other hand, it could be a summary of many years of experience doing something you do well. Chances are, if you've ever struggled with something in the FreeBSD arena, someone else has too,

I'm here to tell you
that writing is an
interesting, fun way
to learn more about
FreeBSD

or they will in the future; either way, they will benefit from hearing your side of the story. Your article might be the one that moves them forward, opens a door, gets them back on track. It might be through learning a new skill or a different way of thinking about something. I can't tell you the number of times I've received feedback on an article I wrote that helped me too.

Whenever I write an article, I am careful to detail—in words—the very same steps of the processes that I use to get my FreeBSD machine to do what I want it to do. At the outset, I write in pieces and often with the help of virtual sticky notes or a basic document with a few spaces between ideas. As I continue to develop it, I add to my initial thoughts and fill in the spaces and notes with bits of information I want to convey and/or the commands I use. Before long, I've built up a respectable amount of text and examples that I then link together into a story. This is just an example of how I work; other writers may work differently. But all of us compose and edit as best we can before we send it off to the *FreeBSD Journal* to review. The *Journal* staff does a great job of making the article read even better, and you are asked to review the edited version and design, to update your content or make corrections, before publication.

What and how you write is in your hands, the final content and design subject to your approval, but, whether you're new to writing about FreeBSD or someone who's been at it for a while, there are always other hands that can help you along the way. Consider giving it a try!

ROLLER ANGEL spends most of his time helping people learn how to accomplish their goals using technology. He is an avid BSD user and Pythonista who enjoys learning all the amazing ways he can solve issues with BSD and Python. He is a firm believer that people can learn anything they wish to set their minds to. He contributes to open source projects in his spare time and uses BSD and Python to learn more about the computer, to assist with his many side projects, and to automate and manage various aspects of his life.



The FreeBSD Project is looking for

- Programmers • Testers
- Researchers • Tech writers
- Anyone who wants to get involved

Find out more by

Checking out our website

freebsd.org/projects/newbies.html

Downloading the Software

freebsd.org/where.html

We're a welcoming community looking for people like you to help continue developing this robust operating system. Join us!

Already involved?

Don't forget to check out the latest grant opportunities at freebsd.foundation.org

Help Create the Future. Join the FreeBSD Project!

FreeBSD is internationally recognized as an innovative leader in providing a high-performance, secure, and stable operating system.

Not only is FreeBSD easy to install, but it runs a huge number of applications, offers powerful solutions, and cutting edge features. The best part? It's FREE of charge and comes with full source code.

Did you know that working with a mature, open source project is an excellent way to gain new skills, network with other professionals, and differentiate yourself in a competitive job market? Don't miss this opportunity to work with a diverse and committed community bringing about a better world powered by FreeBSD.

The FreeBSD Community is proudly supported by



Why do you use FreeBSD?



In June 2020, the FreeBSD Core Team surveyed the FreeBSD community. Of the many questions asked, one that is of interest to the folks just learning about FreeBSD or new to the community is “Why do you use FreeBSD?” Below you’ll find just a small sample of the responses, directly from the users themselves. If you’ve ever wondered why someone might choose FreeBSD, take a look. We think you’ll like what you see. **Anne Dickison**

“FreeBSD allowed me to learn skills that have remained valid over a decade. Being able to use muscle memory for basics let me compound my understanding of the system and confidence in knowing and trusting the substrate on which my applications stand.”

“I use FreeBSD because it provides me with a fast, secure, and fully customizable server platform to handle whatever I throw at it.”

“[I]t’s the Swiss army knife of OSes. If you’ve got a problem that needs to be solved in a hurry, you can almost always count on being able to roll out a FreeBSD-based solution in time.”

“The system, the documentation, the community and the high level of dedication within is 100% in alignment with my way of working and communicating. It is just an amazing “friendship” that has lasted for more than 20 years.”

“I came because of systemd and stayed because of the awesome community; when I started a company FreeBSD was the only operating system I could use for my application.”

“I use FreeBSD because of the consistent user/administrator experience and the well-reasoned technical decisions of the Project.”

“I use FreeBSD for being a system that I can trust to run for a long time. Upgrading releases is a forethought for the Project and not an afterthought. So when 13.0 comes out, I know I can cleanly upgrade to it, and I have tools that let me roll back if something doesn’t go right. And while running that version of FreeBSD, I can trust that it will continue to run reliably.”

“I use FreeBSD because it’s stable, easy to manage, and consistent!”

“Because it’s of great quality. Because it works “out-of-the-box” if you have studied it for the first time. Because it provides all I need to do my daily job at home—browsing the Internet, listening to the music, watching movies or YouTube, coding, serving as a NAS appliance, gaming either natively or using Linux games, even gaming in wine if needed. Because FreeBSD doesn’t change core internals like Linux distros do with their systemd or pulseaudio. Because everything in the base system just works and does it repeatedly and without strange problems like with mentioned pulseaudio. Because community is helpful and knowledgeable. Because FreeBSD opened a world of Unix for me back in 2006. Thank you, guys.”

“FreeBSD offers a unique combination of a rock-solid open-source operating system with high standards for security and performance that integrates lightweight virtualization (jails) as well as the most advanced storage subsystem (ZFS).”

“It “just works.” I can use it to do what I want—WRT network management, online activities, monitoring, media handling and development. There are no limits—if something’s not there, it can be created... there are no secrets!”

“FreeBSD is elegant, robust, and efficient. The configuration files are concise and well placed, and the filesystem and security options are compelling. FreeBSD is a pleasure to work with and use.”

"I like FreeBSD because it is an OS, not a distribution. There is a strong vision attached to it."

"It is single entity controlled distribution, well documented and with a rich set of available end-user software installable from centralized repository. It is fast and reliable."

"FreeBSD is the most-well-thought-out and easiest-to-learn operating system I have ever used. ZFS is a first class citizen, and I can install a system on top of it with no additional effort. The pkg command is a shining example of user friendliness. Having a base system separate from my installed packages keeps things sane and organized. I love it."

"FreeBSD is the most dependable, consistent, available, and cost-effective computing environment, period."

"I switched from Linux to FreeBSD for most applications because Linux feels like it is constantly chasing the "next shiny thing" and has forgotten most of its UNIX heritage. With FreeBSD, system administration is fun again, my files are safe, and my heart is at ease."

"I like the spirit of the community, the knowledgeable people, the good documentation—and that it just isn't Linux. I wouldn't want it any other way."

"I have been using FreeBSD from the very beginnings, and in all those years, FreeBSD has never let me down."

"FreeBSD philosophy with clean and simple, rolling release, stable, open and free. I like BSD* and open source."

"I use FreeBSD because of its stability, great documentation, and security. FreeBSD is used as a key part of my workflow for storage (FreeNAS), networking (both pfSense and iocage+ipfw), and my Nextcloud deployments."

“Stability and security. It is far more stable than Linux systems I have run over the same period.”

“Development not conducted by corporations, I like the democratic way you do things; the performance is also really good. The zfs support, pf as firewall and jails. I love jails.”

“As an operating system, it provides a sane and stable experience. Changes are only made when something is broken or it makes a measurable improvement in the experience.”

“FreeBSD is the best OS experience I ever had. I love the system architecture. FreeBSD mailing list and forums is the best place to get answers, and to learn anything about FreeBSD. Thank you so much.”

“Control, simplicity, and documentation. FreeBSD allows me to actually control how my systems work, all the way down to the OS. On an old laptop with an ACPI table that didn’t advertise the BST properly, it was extremely simple to patch the driver and acpitools to account for the missing DWORDs; and it was easy BECAUSE of the simplicity and documentation.”

“It’s a great and rock-stable OS, well designed with a great documentation.”

“I use FreeBSD because it’s BSD-derived OS that’s easy to configure, uses a sane init system, and has native ZFS support. Jails and bhyve are great, too.”

“Performance. Last week, a prototype of our next-generation CDN server delivered 374Gb/s of real customer traffic over hundreds of thousands of TLS-encrypted sessions from a single machine.”

“I can build the OS I want and need.”

“A real free basically UNIX operating system with power, performance, and security that just gets the job done.”

“FreeBSD lets us implement research prototypes and share them with anyone with any (or no) restrictions.”

“Freedom. Simplicity. Reliability. Good documentation. Because it inspires confidence.”

“I believe FreeBSD has very smart design and it must be used for complicated solutions that require full control over system resources.”

“In one word: consistency. Rather than having to learn a new system with each major release, FreeBSD allows me to focus on maintaining the system and doing other things more important.”

“FreeBSD allows me to see the direct connection between source code and final product. This is what open source is all about.”

“FreeBSD is a versatile and proven tool that never presents an insurmountable problem, or a baffling mystery.”

“I’m able to spend less time doing sysadmin work and more time focusing on the applications that I get paid to deliver. Configuration via simple text files, a well-designed and strictly applied file hierarchy, and easy-to-understand scripts to control services (no giant blob of dubious quality code like systemd) all make my life easier.”

“FreeBSD is better organized and it’s easy to use. It has clean structure. Jail support allows us to replace Citrix XEN in many cases. Its license is very liberal.”

“FreeBSD support is a competitive advantage; FreeBSD keeps us honest when it comes to portability; and FreeBSD has always been a helpful and engaging community. I appreciate the level of professionalism with which FreeBSD conducts itself.”

Support FreeBSD®



Donate to the Foundation!

You already know that FreeBSD is an internationally recognized leader in providing a high-performance, secure, and stable operating system. It's because of you. Your donations have a direct impact on the Project.

Please consider making a gift to support FreeBSD for the coming year. It's only with your help that we can continue and increase our support to make FreeBSD the high-performance, secure, and reliable OS you know and love!

Your investment will help:

- Funding Projects to Advance FreeBSD
- Increasing Our FreeBSD Advocacy and Marketing Efforts
- Providing Additional Conference Resources and Travel Grants
- Continued Development of the FreeBSD Journal
- Protecting FreeBSD IP and Providing Legal Support to the Project
- Purchasing Hardware to Build and Improve FreeBSD Project Infrastructure

Making a donation is quick and easy.
freebsd.foundation.org/donate





Events Calendar

BSD Events taking place through November 2020

BY ANNE DICKISON



OpenFEST 2020

November 7 & 8

[VIRTUAL](#)

OpenFest is the largest conference in Bulgaria dedicated to free and open-source software. Its purpose is to advocate for projects created with free and open software and to provide a venue for exchanging ideas and best practices in the field. This year's event will be held online. The FreeBSD Foundation Executive Director, Deb Goodkin, will be presenting an Introduction to FreeBSD.



FreeBSD Vendor Summit

November 11–13

[VIRTUAL](#)

The 2020 FreeBSD Vendor Summit will consist of virtual, half-day sessions. It's free to attend, but you must register with the Eventbrite system to gain access to the meeting room. In addition to vendor talks, there will be discussion sessions.

FreeBSD Fridays

<https://freebsd.foundation.org/freebsd-fridays/>

November 6: Introduction to RISC-V on FreeBSD—*Mitchell Horne*

Mitchell Horne will discuss the past, present, and future of FreeBSD's support for the RISC-V CPU architecture.

November 20: Introduction to D-Trace

Join Mark Johnston as he takes you through an Introduction to DTrace on FreeBSD.

Past FreeBSD Fridays sessions are available at: <https://freebsd.foundation.org/freebsd-fridays/>

FreeBSD Office Hours

<https://wiki.freebsd.org/OfficeHours>

Join members of the FreeBSD community for FreeBSD Office Hours. From general Q&A to topic-based demos and tutorials, Office Hours is a great way to get answers to your FreeBSD-related questions.

Past episodes can be found at the FreeBSD YouTube Channel

<https://www.youtube.com/c/FreeBSDProject>